Departamento de Ingeniería de Sistemas y Automática

PROYECTO FIN DE CARRERA

CREACIÓN DE MAPAS VISUALES PANORÁMICOS MEDIANTE TÉCNICAS DE VISIÓN ARTIFICIAL

Autor: Fernando Ortiz Renilla
Tutor: Jorge García Bueno

Leganés, Febrero 2012

Título: Creación de Mapas Visuales Panorámicos mediante Técnicas de Visión Artificial
Autor: Fernando Ortiz Renilla
Director: Jorge García Bueno

EL TRIBUNAL

Presidente: Alejandro Martín Clemente

Vocal: Marcos Rodríguez Millán

Secretario: David Álvarez Sánchez

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 3 de Julio de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

Tras un largo camino recorrido es el momento de hacer balance y agradecer de la manera más sincera a todas aquellas personas que han contribuido de alguna manera en la realización de este proyecto.

En primer lugar quiero agradecer a mi tutor Jorge García por sus consejos y explicaciones en los momentos de dudas que me surgieron durante la realización del proyecto. Nada de esto hubiera sido posible sin su inestimable ayuda.

Quiero agradecer a mis compañeros de clase por todos estos años que hemos pasado juntos, y en especial a Alberto González, cuya ayuda he tenido siempre disponible a lo largo de toda la carrera.

También quiero dar las gracias a mi familia, y en especial a mis padres, por todo el apoyo que me han prestado, tanto en el sentido psicológico como en el económico.

Un agradecimiento especial para Sergio Martínez, encargado de guiarme en los principales temas burocráticos. Pese a que comenzamos la universidad juntos, ésta se ha empeñado en separarnos, pero afortunadamente nos une una amistad que difícilmente podrá romperse.

Gracias también a mis amigos de toda la vida, que durante los últimos meses han visto como el ordenador portátil se convertía para mí en un amigo como cualquiera de ellos. Sin vosotros no sería lo que soy.

Y finalmente me gustaría agradecer a Ana Recuero, una persona muy especial para mí, que me ayudó con la revisión del texto y la elección del formato de la memoria y los CDs, además de apoyarme y animarme todos y cada uno de los días que empleé en la realización del proyecto.

A todos ellos, muchas gracias.

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO 2: ESTADO DEL ARTE

CAPÍTULO 3: HERRAMIENTAS Y PLATAFORMA UTILIZADA

CAPÍTULO 4: METODOLOGÍA EMPLEADA

CAPÍTULO 5: CONCLUSIONES

ANEXO 1: INFORME DETALLADO DEL ESTUDIO DEL TIEMPO DE EJECUCIÓN

ÍNDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1.1. INTRODUCCIÓN

Desde los comienzos de la informática y respondiendo a distintas motivaciones el hombre ha querido simular comportamientos humanos mediante el uso de computadoras. Acciones que para una persona son relativamente sencillas de realizar, como por ejemplo memorizar la cara de otra persona o distinguir un perro de un gato, son bastante complicadas de implementar en un sistema informático. Y, en el caso contrario, una persona normal es incapaz de realizar miles de cálculos matemáticos en apenas unos segundos, algo fácilmente realizable con una computadora.

Una de las principales capacidades del ser humano en la que se trabaja para trasladar al entorno informático es el sentido de la vista, mediante la ciencia de la visión por computador. Son innumerables las aplicaciones que pueden obtenerse a través de la computarización de imágenes en tiempo real, siendo ésta cada vez más avanzada y funcional.

La emulación de la visión humana mediante un ordenador resulta muy complicada, por lo que se suele dividir el problema de la visión por computador en distintas fases, cuya solución resulta más asequible. Cada una de estas fases puede relacionarse con otras disciplinas y especialidades para simplificar cada parte del proceso. Algunos ejemplos son el procesamiento de imágenes, el seguimiento de objetos o el reconocimiento de patrones.

Una de las aplicaciones que más demandan el uso de visión por computador es la de la robótica, en la cual se enmarca este proyecto. Los robots necesitan cámaras que recojan datos del entorno para procesarlos y poder interactuar con él, por lo que es necesaria no sólo la toma de las imágenes por parte de los sistemas de obtención de imágenes, sino también que posteriormente el controlador del robot realice diversos procesos y tratamientos con ellos. Estos procesos proporcionarán información acerca de los distintos aspectos de la imagen recogida y facilitarán el uso posterior de los datos obtenidos.

El objetivo de este proyecto es crear una aplicación para proporcionar al robot una visión panorámica del entorno que le rodea. Mientras que para una persona es fácil obtener una imagen del entorno que le rodea mediante un vistazo rápido, para un robot es una tarea más complicada, debido a las restricciones de la cámara y de movimientos.

1.2. OBJETIVOS INICIALES DEL PROYECTO

El proyecto tiene como misión crear un programa en C ó C++ que sea capaz de formar mapas panorámicos mediante la fusión de imágenes correlativas tomadas directamente a través de una cámara. Estos mapas posteriormente podrán ser utilizados para diversas aplicaciones como la detección de personas u objetos.

Para ello, se usarán descriptores SIFT/SURF para determinar la posición relativa de las imágenes y así poder mezclar los frames consecutivos para ulteriormente generar el mapa panorámico. El tratamiento necesario de las imágenes se realizará mediante las bibliotecas OpenCV trabajando sobre Linux.

El proyecto formará parte de la primera fase de un sistema de localización de personas y objetos diseñado para su futura implementación en robots.

1.3. FASES DE DESARROLLO

Para el desarrollo del proyecto se ha seguido el siguiente proceso:

- Búsqueda de información y documentación acerca de los algoritmos SIFT y SURF, así como de la librería de OpenCV.
- Adaptación de los recursos de hardware mediante la instalación del sistema operativo Ubuntu y la librería OpenCV, además de otros paquetes necesarios para el desarrollo de la aplicación.
- Familiarización y realización de los primeros programas con cámara web y OpenCV.
- Creación de una primera versión del programa en el que las imágenes se adquieren anteriormente a la ejecución, para después insertarse en el programa a través de archivo.
- Sustitución en el programa de las rutinas de las imágenes insertadas por archivo por las necesarias para la captura directamente desde la cámara web.
- Comprobación de los resultados y mejoras.

1.4. ESTRUCTURA DE LA MEMORIA

Con el fin de facilitar la lectura de la memoria, se incluye a continuación un breve resumen de cada capítulo:

Capítulo 1: Introducción

En este apartado se explican de forma introductoria los objetivos y características principales del proyecto, así como un resumen del contenido de la memoria y del proceso realizado.

Capítulo 2: Estado del arte

En este capítulo se dan a conocer las bases teóricas sobre las que se fundamenta el proyecto. No sólo se describen las técnicas que se han usado para el desarrollo del proyecto, también aquellas técnicas que por su importancia y utilización dentro del campo de la visión artificial he creído conveniente destacar.

Capítulo 3: Herramientas y plataforma utilizada

Explicación detallada de cada una de las herramientas utilizadas para la creación del proyecto. Se incluyen referencias tanto al hardware utilizado como al software, con especial interés en la librería OpenCV, de la que se enumeran las funciones utilizadas en el programa.

Capítulo 4: Metodología empleada

Capítulo dedicado a exponer los métodos y técnicas que se han usado. Se recoge el funcionamiento general del programa y se explica en profundidad cada una de las fases que realiza.

Capítulo 5: Conclusiones

Se recogen los resultados experimentales obtenidos así como estudios acerca del movimiento adecuado de la cámara y del tiempo de ejecución del programa. También se incluyen una serie de posibles mejoras aplicables al programa.

Anexo 1: Informe detallado del estudio de los tiempos de ejecución

En este anexo se recopilan todos los datos de tiempos recogidos de la creación de las panorámicas usadas para el estudio de los tiempos de ejecución.

Anexo 2: Diagrama de flujo

Se presenta el diagrama de flujo del programa, que nos facilita la compresión de funcionamiento del programa exponiéndolo como un dibujo.

Anexo 3: Programa

Transcripción íntegra del programa, con numerosos comentarios explicativos de su funcionamiento. Se han coloreado las variables y palabras reservadas del lenguaje tal y como lo hace el editor de Ubuntu para facilitar su lectura.

CAPÍTULO 2

ESTADO DEL ARTE

2.1.- VISIÓN POR COMPUTADOR

La visión por computador es una disciplina tan amplia que existen infinidad de libros dedicados a las diversas áreas que abarca. Muchos de estos contenidos son superfluos para la realización del proyecto, por lo que en este apartado se expone un breve resumen de las nociones básicas acerca de la visión por computador.

2.1.1 CARACTERÍSTICAS DE UN SISTEMA DE VISIÓN

La visión por computador actualmente comprende tanto la obtención como la caracterización e interpretación de las imágenes. Por lo tanto, un sistema de visión artificial está compuesto por un sensor óptico, para capturar la imagen, y de un computador que almacene las imágenes y ejecute los algoritmos de pre-procesado, segmentación y reconocimiento.

2.1.1.1 FASES DE UN SISTEMA DE VISIÓN

En un sistema de visión por computador actual se pueden distinguir seis etapas o fases, desde los procesos de obtención de la imagen que se desea capturar hasta la etapa final de interpretación de los datos obtenidos de ella, pasando por todas las fases necesarias para la obtención de esos datos. Estas fases son:

- **Captación:** Proceso a través del cual se obtiene una imagen visual desde una cámara.
- **Pre-procesado:** Conjunto de técnicas que facilitan el procesado posterior. Incluye técnicas como la reducción de ruido y realce de detalles.
- **Segmentación:** Proceso que se encarga de dividir una imagen en regiones que puedan ser de nuestro interés.
- **Descripción:** Proceso mediante el cual se obtienen características convenientes para diferenciar un tipo de objeto de otro.
- **Reconocimiento:** Proceso por el que podemos identificar los objetos de una escena.
- **Interpretación:** Tareas que proporcionan un significado a un conjunto de objetos reconocidos en una imagen.

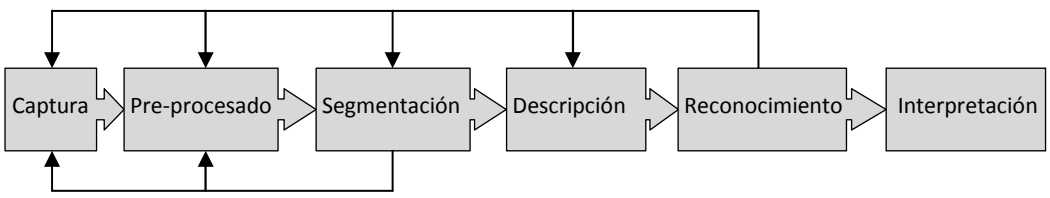


Figura 2.1.- Fases de un sistema de visión por computador.

2.1.1.2. NIVELES DE VISIÓN

A su vez la visión por computador se puede dividir en distintos niveles, dependiendo de la fase o fases en las que nos interese trabajar. Son:

- **Low-level vision:** comprende la captación y el pre-procesamiento. Ejecuta algoritmos típicamente de filtrado, restauración de la imagen, realce, extracción de contornos, etc.
- **Medium-level vision:** comprende la segmentación, descripción y reconocimiento, con algoritmos típicamente de extracción de características, reconocimiento de forma y etiquetado de éstas.
- **High-level vision:** comprende la fase de interpretación, normalmente estos algoritmos se refieren a la interpretación de los datos generalmente mediante procedimientos típicos de la Inteligencia Artificial para acceso a bases de datos, búsquedas, razonamientos aproximados, etc.

2.1.2 CAPTURA Y REPRESENTACIÓN DE LA IMAGEN

El proceso que sigue la imagen para ser digitalizada es relativamente simple. Todas las imágenes digitalizadas se representan mediante el uso de píxeles, que consisten en elementos individuales que contienen información de esa determinada región de la imagen. En imágenes a color cada píxel contiene la información de cada una de las componentes de una base de color; mientras que en una imagen en escala de grises corresponde a la información del brillo.

2.1.2.1 CAPTURA Y DIGITALIZACIÓN

La técnica para la captura de imágenes consiste en la transformación de unos datos continuos (el espacio y la luz) en otros discretos. Este proceso consta de dos etapas: primero se capturan los datos reales que queremos representar (captura), y posteriormente se interpretan y traducen a datos digitales (digitalización)

La captura se puede realizar mediante dos tipos de dispositivos:

- **Pasivos:** Están basados en el principio de cámara oscura, es decir, recogen la luz de la escena y la proyectan sobre un plano bidimensional, como se muestra en la Figura 2.2. Unos buenos ejemplos son las cámaras de fotos y vídeo.

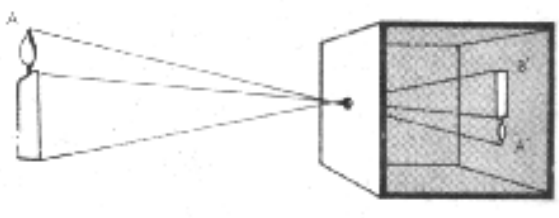


Figura 2.2.- Principio de cámara oscura.

- **Activos:** Utilizan una fuente de luz para recoger datos de la escena u objeto, por lo tanto constan de un elemento emisor y otro receptor. Un ejemplo de este tipo de dispositivo sería un escáner, cuyo funcionamiento se expone en la Figura 2.3.

Figura 2.3.- Esquema de funcionamiento de un escáner.

En el proceso de digitalización se distinguen dos subprocesos:

- **Sampling (Muestreo):** Como su propio nombre indica consiste en tomar muestras de la señal continua a una frecuencia determinada, convirtiendo la imagen continua en una matriz de píxeles. Cuantas más muestras obtengamos del objeto por unidad de espacio, más resolución tendrá la imagen digitalizada.

- **Quantization (Cuantificación):** Consiste en la discretización de los posibles valores de cada píxel. Suelen usarse potencias de 2 para facilitar el almacenamiento de los datos.

2.1.2.2 ESTRUCTURA DE LOS DATOS DE IMAGEN

El ordenador almacena los datos de la imagen en forma de ficheros. Estos ficheros están formados por un mapa de bits y una cabecera que describe sus características.

Habitualmente las imágenes se comprimen debido al gran tamaño de los datos que contienen. La compresión consiste en reducir el tamaño de la imagen eliminando la información redundante, aunque se disminuya la calidad de la imagen (ver Figura 2.4).

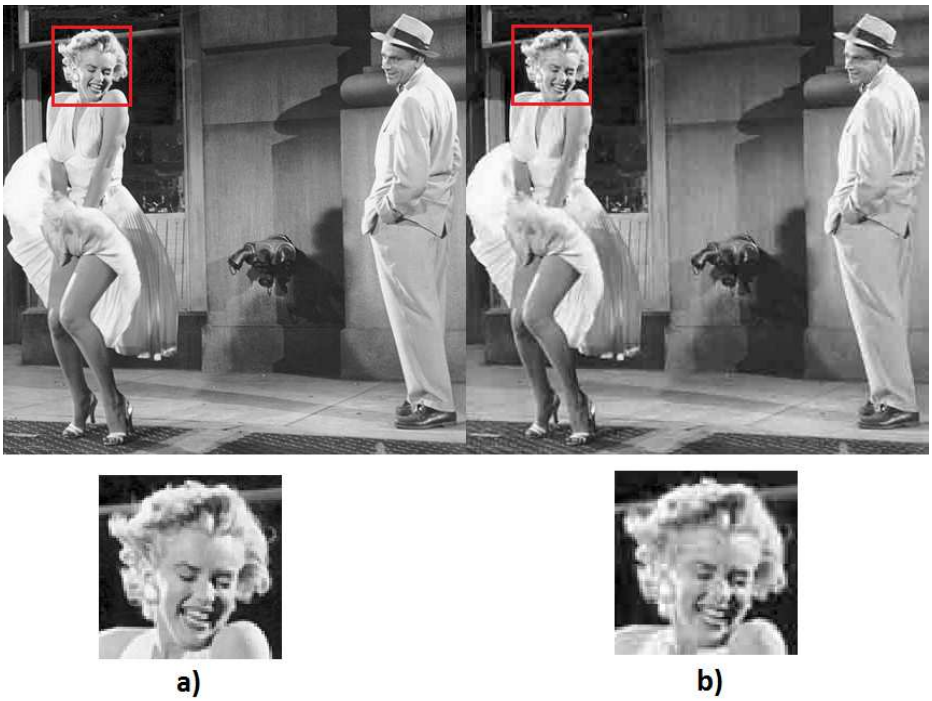


Figura 2.4.- Efecto de la compresión JPEG, donde se aprecian los defectos que introduce: a) Imagen sin comprimir, 600KB b) Imagen comprimida, 40KB

2.1.3 PROCESAMIENTO DE IMÁGENES

En este capítulo se tratan las operaciones básicas de tratamiento de imágenes. Se describirán tres tipos de operaciones: operaciones básicas posibles entre píxeles, operaciones sobre el histograma y operaciones de filtrado.

2.1.3.1 OPERACIONES BÁSICAS ENTRE PÍXELES

Las operaciones directas sobre píxeles se pueden clasificar en operaciones aritméticas y lógicas por una parte, y operaciones geométricas por otra.

Operaciones aritméticas y lógicas

Operaciones lógicas:

- **Conjunción:** Operación lógica AND entre los bits de dos imágenes. Se usa para borrar píxeles en una imagen.
- **Disyunción:** Operación lógica OR entre los bits de dos imágenes. Se usa para añadir píxeles a una imagen.
- **Negación:** Inversión de los bits que forman una imagen. Se usa para obtener el negativo de una imagen.

Operaciones aritméticas:

- **Suma:** Suma de los valores de los píxeles de dos imágenes.

- **Resta:** Resta de los valores de los píxeles de dos imágenes.
- **Multiplicación:** Multiplicación de los valores de los píxeles de una imagen por los de otra. Se usa para añadir textura a una imagen.
- **División:** División de los valores de los píxeles de una imagen entre los de otra.

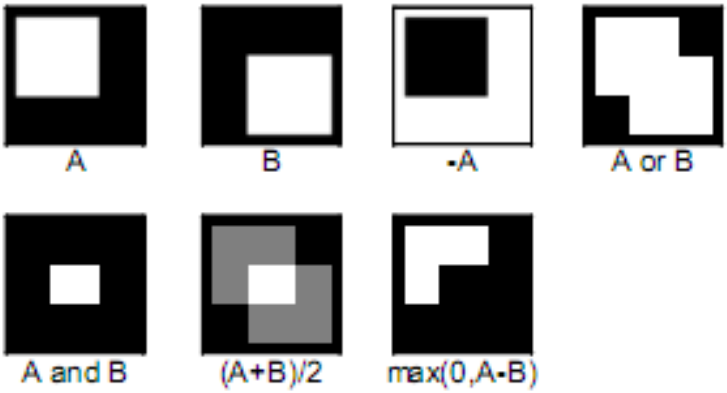


Figura 2.5.- Ejemplos de operaciones aritméticas y lógicas entre píxeles. Los píxeles en negro corresponden a bits a 0, los blancos a bits a 255.

Operaciones geométricas

Todas las transformaciones se pueden tratar mediante multiplicación de matrices. Las operaciones geométricas más usuales son:

- **Traslación:** Movimiento de los píxeles de una imagen según un vector de movimiento.
- **Escalado:** Cambio del tamaño de una imagen.
- **Rotación:** Giro de los píxeles de una imagen en torno al origen de coordenadas.

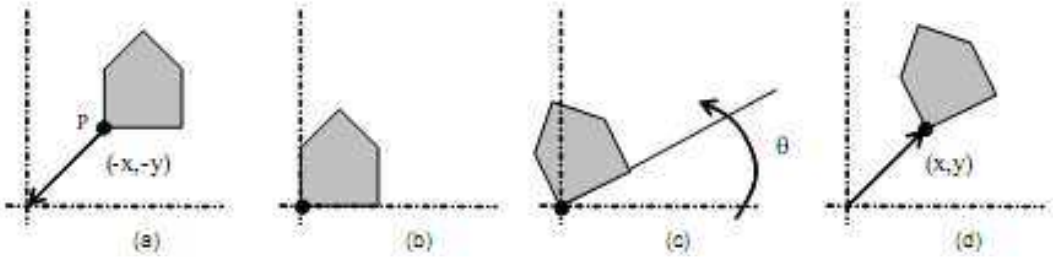


Figura 2.6.- Ejemplo de rotación: (a) Imagen original que se desea rotar en torno al punto P de coordenadas (x,y) (b) resultado de la primera traslación (c) resultado del giro (d) resultado final después de la última traslación.

2.1.3.2 MODIFICACIÓN DEL HISTOGRAMA

El histograma de una imagen es un diagrama de barras que representa los datos que contiene la imagen. Cada barra tiene una altura proporcional al número de píxeles que hay para un nivel de cuantificación dado.

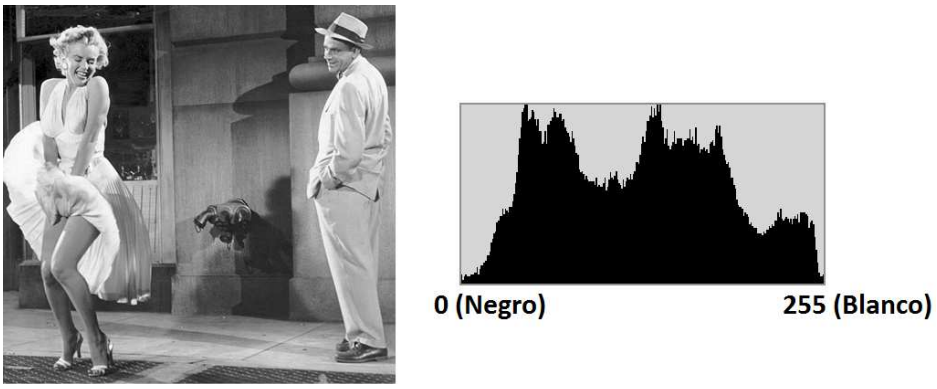


Figura 2.7.- Imagen en niveles de gris y su correspondiente histograma.

El histograma de una imagen en niveles de gris proporciona información sobre el número de píxeles que hay para cada nivel de intensidad. En imágenes en color RGB se usan 3 histogramas, uno por cada componente de color.

Modificación del contraste

Es posible ajustar parámetros de la imagen modificando su histograma con el objetivo de mejorar la imagen para facilitar los tratamientos posteriores. Una de estas posibles modificaciones es la del contraste de la imagen. El contraste es la diferencia relativa en intensidad entre un punto de una imagen y sus alrededores.

Cuando reducimos el contraste el histograma de la imagen se contrae, eliminando las zonas de valores extremos y proporcionando a la imagen una apariencia apagada y grisácea. Por el contrario, cuando aumentamos el contraste, el histograma se expande, llegando incluso a quedar algunas zonas saturadas (Figura 2.9)

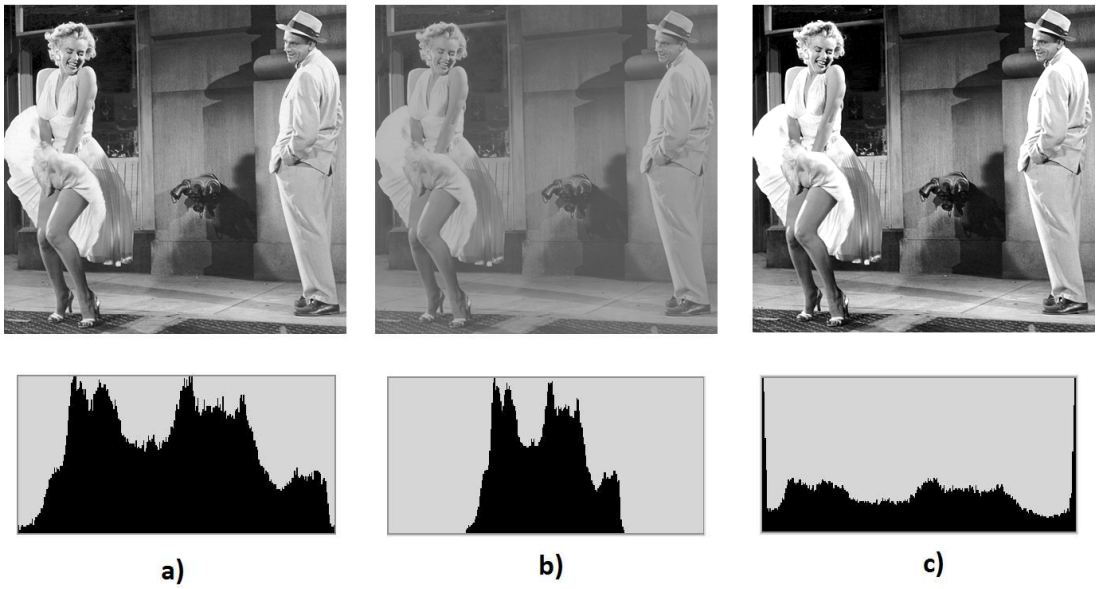


Figura 2.8.- Transformaciones del histograma: a) Imagen original. b) Disminución de contraste. c) Aumento de contraste.

Ecualizado del histograma

El proceso de ecualizado tiene por objetivo obtener un nuevo histograma, a partir del histograma original, con una distribución uniforme de los diferentes niveles de intensidad.

Al transformar cualquier distribución continua en una distribución uniforme se está maximizando la cantidad de información que contiene. Y aunque ya se ha dicho que en el caso discreto es imposible aumentar la cantidad de información, el ecualizado del histograma puede mejorar la calidad visual de imágenes parcialmente saturadas. Este efecto se debe a que se cambian los valores de intensidad de las zonas saturadas, en las que originalmente existen objetos que no se distinguen adecuadamente al inspeccionar visualmente la imagen.

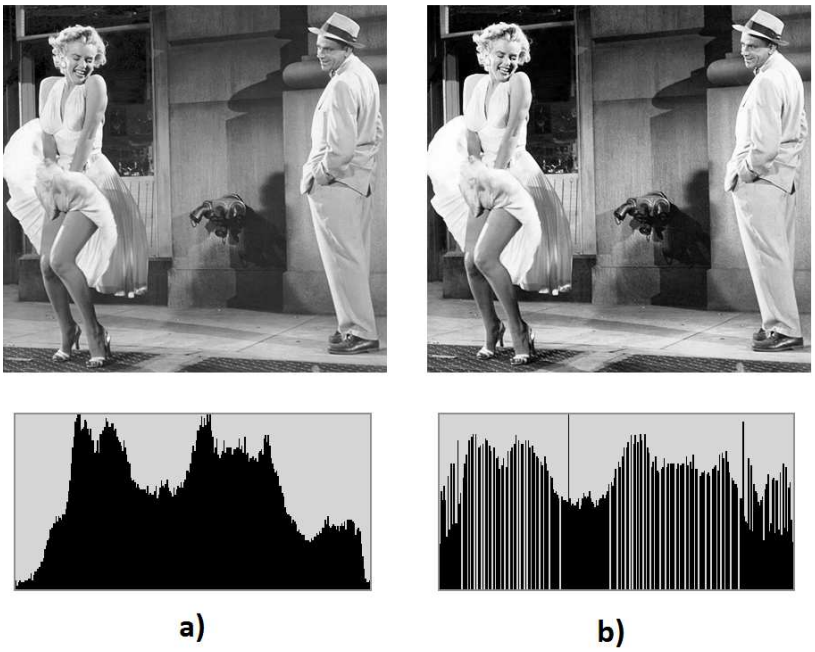


Figura 2.9.- Ecualizado del histograma: a) Imagen original. b) Ecualizado del histograma.

2.1.3.2 FILTRADO

Otra operación posible sobre la imagen es la de filtrado. Es una operación que se realiza en el dominio del espacio. Existen multitud de técnicas de filtrado con diferentes aplicaciones, pero debido a que es innecesario para nuestro objetivo, sólo nombraremos alguna de ellas:

- **Filtros de suavizado:** Se basa en el promediado de los píxeles adyacentes al píxel que se evalúa. El efecto es un desenfoque de la imagen.
- **Filtros de obtención de contornos:** Se basa la derivada direccional de una función permite conocer cómo se producen los cambios en una dirección determinada.
- **Filtros de laplaciana:** Se basa en el operador laplaciano. También se usa para la detección de contornos.

Figura 2.10.- Ejemplo de filtrado suavizado. a) Imagen original. b) Imagen suavizada

2.1.4 ALGORITMOS SIFT Y SURF

La detección de objetos en una imagen es el procedimiento mediante el cual se ubican en la imagen los objetos de interés. Constituye una tarea complicada debido a la cantidad de variables que pueden afectar a la apariencia del objeto.

Existen diversos métodos y algoritmos que se encargan de esta tarea, aunque nosotros usaremos los llamados SIFT y su derivado SURF, cuyos principios básicos repasaremos brevemente en este apartado.

2.1.4.1 DESCRIPTOR SIFT

El descriptor SIFT (Scale-Invariant Feature Transform) fue descrito por David G. Lowe y tiene como objetivo la detección y descripción de las características locales de una imagen para la detección de objetos y patrones. Se basa en la detección puntos de interés en la imagen para crear una "descripción" del objeto, que se usará para su posterior detección.

Para que la detección sea fiable, es importante que las características extraídas de la imagen sean detectables incluso con cambios en la escala, ruido e iluminación. Estos puntos de interés suelen encontrarse en regiones de alto contraste, como los bordes del objeto.

El algoritmo SIFT se compone principalmente de cuatro etapas que se describen siguiendo la implementación de Lowe:

1. **Detección de extremos en el espacio-escala:** El primer paso es la búsqueda de puntos en la imagen que puedan ser puntos clave. Se realiza usando diferencias de funciones gaussianas para identificar puntos interesantes que sean invariantes a la escala y la orientación.

2. **Localización de los puntos clave:** De los puntos obtenidos en el apartado anterior se determinan la localización y la escala de los mismos, de los cuales se seleccionan los puntos clave basándose en la media de la estabilidad de los mismos.

3. **Asignación de la orientación:** A cada localización del punto clave se le asigna una o más orientaciones, basadas en las orientaciones de los gradientes locales de la imagen.

4. **Descriptores de los puntos clave:** Los gradientes locales se miden y se transforman en una representación que permite importantes niveles de la distorsión de la forma local y el cambio de la iluminación.

Detección en el espacio-escala

La primera de las etapas tiene como objetivo obtener puntos candidatos de la imagen que puedan ser identificados de forma repetida bajo diferentes vistas del mismo objeto. El descriptor SIFT es construido a partir del espacio-escala Gaussiano de la imagen original, en el cual se pueden detectar de manera efectiva las posiciones de los puntos claves, invariantes a cambios de escala de la imagen. El espacio-escala Gaussiano de una imagen $L(x, y, \sigma)$ es definido como la convolución de funciones 2D Gaussianas $G(x, y, \sigma)$ de diferentes valores σ con la imagen original $I(x, y)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

siendo (x, y) las coordenadas espaciales y σ el factor de escala.

El algoritmo utiliza la función DoG (Diferencia de Gaussiana) que se forma a partir de la derivada escalar de la Gaussiana escalada espacialmente. Esta función DoG $D(x, y, \sigma)$ se obtiene mediante la sustracción de escalas posteriores en cada octava:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

donde k es una constante multiplicativa del factor de escala. La función DoG es utilizada por varias razones. En primer lugar porque es una función eficiente en cuanto a coste computacional se refiere: Las imagenes suavizadas $L(x, y, \sigma)$ son calculadas para la descripción de características en el espacio-escala, y por lo tanto, D puede obtenerse como una simple resta.

Al conjunto de las imágenes Gaussianas suavizadas junto con las imágenes DoG se le llama octava. El conjunto de las octavas es construido mediante el muestreo sucesivo de la imagen original por un factor de 2. Cada una de ellas es a su vez dividida en un número entero de subniveles o escalas *s*. Una vez se ha procesado una octava completa, la primera imagen de la siguiente octava se obtiene mediante el muestreo

de la primera de las imágenes de la octava predecesora con un valor de σ del doble respecto a la actual. Esto se traduce en una gran eficiencia del algoritmo para un número de escalas pequeño. El proceso descrito puede verse representado en la Figura 2.11. Es importante tener en cuenta que la imagen original es expandida en el inicio del proceso para crear más puntos de muestreo que en la imagen original, por lo que la imagen resulta duplicada en tamaño antes de construir el primer nivel de la pirámide.

Dado que el espacio-escala $L(x, y, \sigma)$ representa la misma información a diferentes niveles de escala, el modo particular del muestreo permite una reducción de la redundancia. De esta manera se producen s + 3 imágenes por cada una de las octavas y por lo tanto s + 2 DoG imágenes donde se llevará a cabo la búsqueda de extremos. De acuerdo con los resultados de Lowe, es el valor de s = 3 el que mejores resultados consigue.

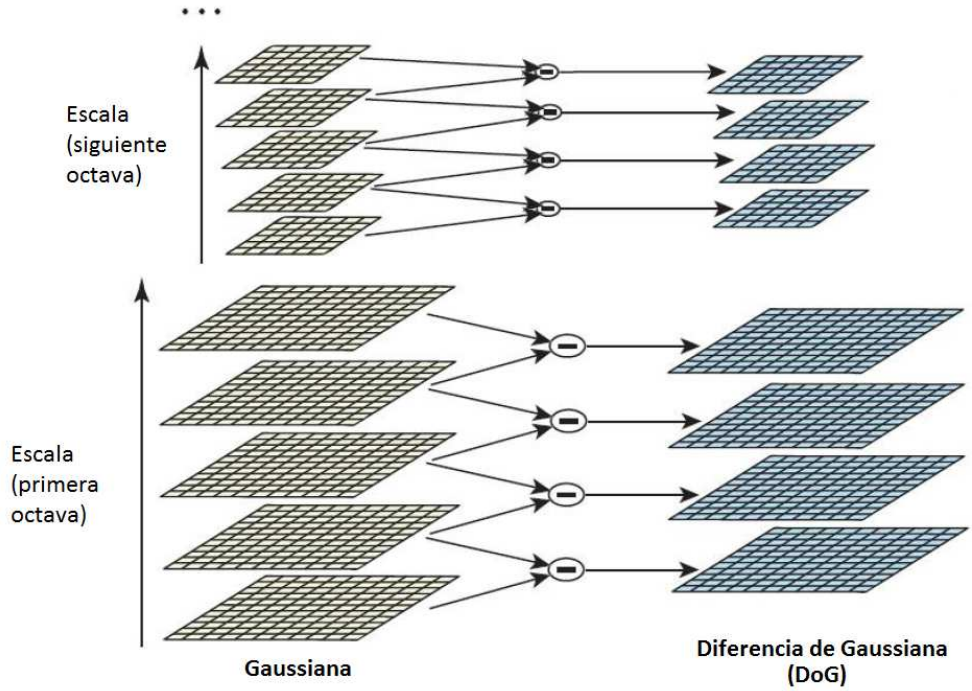


Figura 2.11: Creación del espacio-escala Gaussiano.

Con esto se obtienen 6 imágenes Gaussianas suavizadas y 5 imágenes DoG por cada octava. Respecto del otro parámetro por determinar, σ referente al muestreo de la escala de suavizado, se ha optado por seguir el mismo criterio anterior en base a los resultados de Lowe, donde se determina un valor de σ = 1,6.

Para detectar los máximos y los mínimos locales de cada punto de la imagen $D(x, y, \sigma)$ se compara el valor de éste con el de los puntos vecinos, en concreto, con el de sus 8 vecinos más próximos de la imagen *D* donde se encuentra el punto más los 9 vecinos de cada una de las imágenes *D* de nivel superior e inferior como se muestra en la Figura 2.12. Si el valor resulta ser superior o inferior al de todos sus vecinos, se identifica el punto como máximo o mínimo local respectivamente.

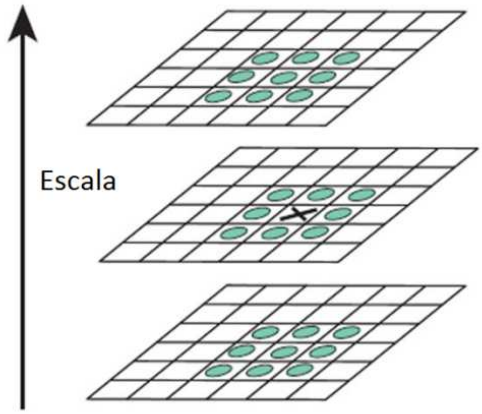


Figura 2.12: Localización de máximos y mínimos locales. Son detectados mediante la comparación de un pixel (marcado con X) con sus 26 vecinos en las regiones de 3x3 de las escalas actual y adyacentes (marcados en azul).

Localización de los puntos clave

Una vez los puntos clave candidatos han sido calculados, en esta segunda etapa se realiza un estudio de su estabilidad. Los puntos no firmemente situados sobre los bordes o aquellos con bajo contraste son bastante vulnerables al ruido y por lo tanto no podrán ser detectados bajo pequeños cambios de iluminación o variación del punto de vista de la imagen.

Para excluirlos, Lowe utiliza los siguientes criterios:

- Para eliminar los puntos con bajo contraste, se aplica un proceso de umbralización por el cual los puntos cuyo valor sea menor que dicho umbral D serán excluidos de la siguiente etapa por no considerarse suficientemente estables. En este proyecto se utiliza el valor de D = 0,03 recomendado por Lowe.

- Los puntos situados sobre bordes de manera difusa, conllevan un alto grado de inestabilidad incluso ante pequeños ruidos. Para llevar a cabo su eliminación, se utiliza la propiedad de la función DoG atendiendo a la gran curvatura que presenta en la dirección paralela al borde y la pequeña curvatura que se observa en la dirección perpendicular. Estas respuestas tan características se pueden estudiar mediante el cálculo de la matriz del Hessiano sobre la localización y escala del punto en estudio:

$$H = \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} \end{bmatrix}$$

donde *D* es la imagen DoG $D(x, y, \sigma)$ respecto de la escala *s*. Las derivadas se calculan mediante la resta del valor de los puntos vecinos. Se puede demostrar que la siguiente desigualdad permite la localización de los puntos en los bordes:

$$\frac{\left(\frac{\partial^2 D}{\partial x^2} + \frac{\partial^2 D}{\partial y^2}\right)^2}{\left(\frac{\partial^2 D}{\partial x^2} \times \frac{\partial^2 D}{\partial x^2}\right) - \left(\frac{\partial^2 D}{\partial x \partial y}\right)^2} < \frac{(r+1)^2}{r}$$

por lo tanto aquellos puntos que no satisfagan dicha desigualdad serán descartados debido a su inestabilidad. Tras descartar los puntos inestables, al resto de puntos clave se les asignará una orientación.

Asignación de la orientación

La característica principal de los puntos SIFT es que éstos son invariantes a una serie de transformaciones sobre las imágenes. La invarianza respecto de la rotación se consigue mediante la asignación a cada uno de los puntos una orientación basada en las propiedades locales de la imagen y representando el descriptor respecto de esta orientación. Para cada uno de los puntos de interés se calcula la magnitud del gradiente, *m*, y su orientación, θ, mediante las siguientes ecuaciones:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \arctan \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}$$

donde *L* representa la imagen gaussiana suavizada cuya escala resulta más próxima a la escala del punto de interés actual.

Respecto de la orientación del gradiente, se crea un histograma con 36 *bins*, cada uno de ellos con una longitud de 10 para cubrir el rango de los 360 posibles. El *bin* cuyo valor es más alto se corresponde con la dirección dominante del gradiente y por lo tanto es elegido como orientación dominante. Sin embargo se ha de tener en cuenta la posibilidad de que exista más de una dirección dominante. Es por ello que cualquier *bin* con un valor de más del 80 % del valor de la magnitud principal se considerará también como dirección dominante. Los puntos que contengan más de una dirección dominante supondrán una mayor estabilidad al mismo. Para una mayor precisión se utiliza una parábola para, mediante la interpolación de los 3 valores más altos del histograma, obtener el valor del pico.

Las orientaciones principales del histograma se asignan al punto de interés para que así el descriptor quede representado respecto de éstos.

Descriptor de los puntos clave

Las etapas anteriores han dotado a los puntos de interés seleccionados de invarianza respecto de la orientación, escalada y localización respecto de la imagen. En esta última etapa se crea un vector de características para cada uno de los puntos de interés que contiene una estadística local de las orientaciones del gradiente de la escala de espacio gaussiano. Se realiza un muestreo de las orientaciones y magnitudes del gradiente de la imagen sobre regiones de 16 x 16 alrededor del punto de interés. Este proceso es similar al de la etapa anterior, donde ahora cada una de las muestras son ponderadas tanto por la magnitud de su gradiente como por una función 3D gaussiana evitando así cambios bruscos en el descriptor ante pequeños cambios en la posición de la ventana y al mismo tiempo asignando menor énfasis a los puntos más alejados del punto de interés. El valor σ de la función gaussiana se fija como 1,5 veces el tamaño de la región de cálculo para el punto de interés.

Se analizan las muestras de cada región de 16 x 16 formando histogramas de orientaciones resumiendo el contenido en sub-regiones de 4 x 4 como se puede ver en la Figura 2.13. Cada uno de los histogramas se compone de 8 *bins*, que almacenan las orientaciones posibles proporcionales a 45 donde la magnitud de cada flecha representa el valor acumulado para cada *bin*. Por lo tanto se obtienen 16 histogramas respecto de las orientaciones de los puntos de cada región para cada uno de los puntos de interés.

Finalmente el descriptor de cada punto de interés está formado por un vector que contiene los valores de las 8 orientaciones de los 4 x 4 histogramas componiendo un vector de características de 4 x 4 x 8 = 128 elementos.

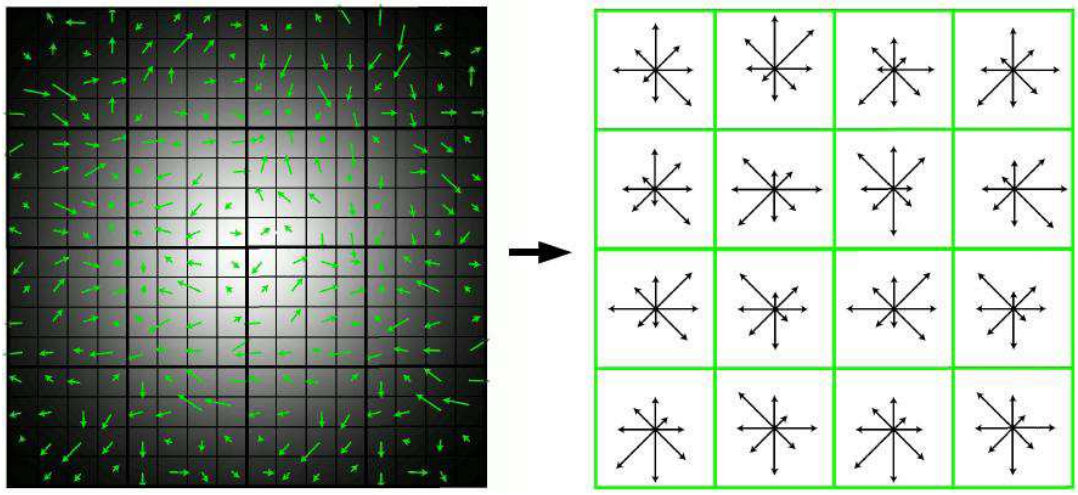


Figura 2.13: Descriptor de los puntos de interés.

De manera añadida, el vector de características es modificado para dotarlo de cierta robustez frente a cambios de iluminación. Los cambios de iluminación afectan en mayor medida a la magnitud del gradiente y no a la orientación, por lo que se busca una representación de esta magnitud que minimice estos efectos. Para ello se lleva a

cabo un proceso de normalización en los que ahora los cambios de contraste (píxeles multiplicados por una constante) quedan neutralizados, mientras que los cambios en la luminosidad (suma de una constante con los píxeles) no afecta a los valores del gradiente que se calcula como diferencias entre píxeles. Si bien esta normalización no confiere invarianza respecto de los cambios de iluminación, sí se consigue paliar los efectos que estos producen.

Finalmente se limita el valor de cada componente de magnitud de gradiente a un valor máximo para que tenga un mayor peso la orientación frente a la magnitud del gradiente. Siguiendo los parámetros de Lowe, el valor del umbral es de 0,2. Luego se vuelve a normalizar de nuevo a una amplitud unidad.

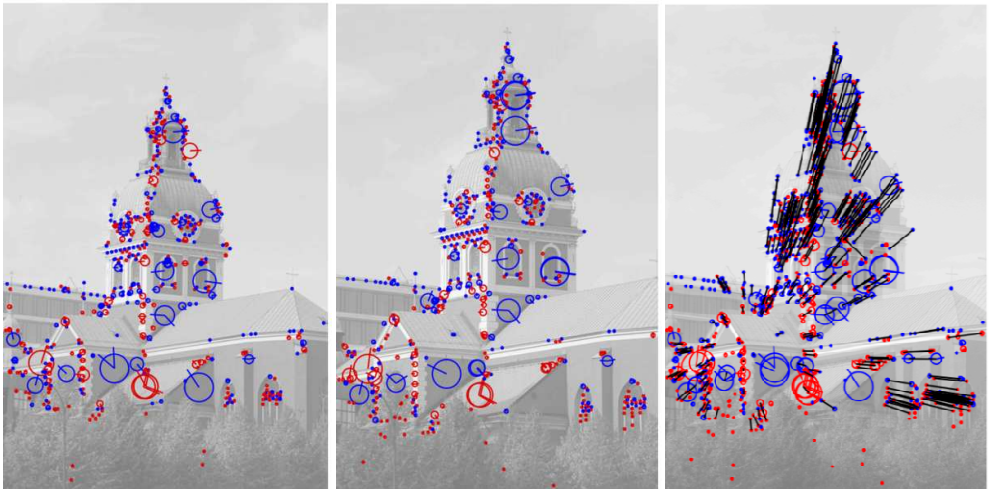


Figura 2.14: Puntos de interés de dos imágenes detectados con el algoritmo SIFT. Las líneas negras unen las parejas de puntos clave.

2.1.4.2 DESCRIPTOR SURF

El descriptor SURF, cuyo acrónimo hace referencia al título, Speeded-Up Robust Features, fue desarrollado por Herbert Bay como un detector de puntos de interés y descriptor robustos. El descriptor SURF guarda cierta similitud con la filosofía del descriptor SIFT, si bien presenta notables diferencias que quedarán patentes con la siguiente exposición sobre su desarrollo. Sin embargo los autores afirman que este detector y descriptor presentan principalmente 2 mejoras resumidas en los siguientes conceptos:

- Velocidad de cálculo considerablemente superior sin ocasionar pérdida del rendimiento.
- Mayor robustez ante posibles transformaciones de la imagen.

Estas mejoras se consiguen mediante la reducción de la dimensionalidad y complejidad en el cálculo de los vectores de características de los puntos de interés obtenidos, mientras continúan siendo suficientemente característicos e igualmente repetitivos.

A continuación se describirán en detalle las etapas para la creación de los descriptores SURF, si bien antes se presentan a modo de resumen previo las diferencias más originales respecto del descriptor SIFT:

- La normalización o longitud de los vectores de características de los puntos de interés es considerablemente menor, concretamente se trata de vectores con una dimensionalidad de 64, lo que supone una reducción de la mitad de la longitud del descriptor SIFT.
- El descriptor SURF utiliza siempre la misma imagen, la original.
- Utiliza el determinante de la matriz Hessiana para calcular tanto la posición como la escala de los puntos de interés.

Detección de puntos de interés

La primera de las etapas del descriptor SURF es análoga a la del descriptor SIFT en cuanto a la detección de puntos de interés se refiere, si bien el procedimiento para su obtención se basa en diferencias sustanciales que se detallan a continuación.

El descriptor SURF hace uso de la matriz Hessiana, más concretamente, del valor del determinante de la matriz, para la localización y la escala de los puntos. El motivo para la utilización de la matriz Hessiana es respaldado por su rendimiento en cuanto a la velocidad de cálculo y a la precisión. Lo realmente novedoso del detector incluido en el descriptor SURF respecto de otros detectores es que no utiliza diferentes medidas para el cálculo de la posición y la escala de los puntos de interés individualmente, sino que utiliza el valor del determinante de la matriz Hessiana en ambos casos. Por lo tanto dado un punto $p = (x, y)$ de la imagen *I*, la matriz Hessiana $H(p, \sigma)$ del punto *p* perteneciente a la escala *σ* se define como:

$$H(p,\sigma) = \begin{bmatrix} L_{xx}(p,\sigma) & L_{xy}(p,\sigma) \\ L_{xy}(p,\sigma) & L_{yy}(p,\sigma) \end{bmatrix}$$

donde $L_{xx}(p, \sigma)$ representa la convolución de la derivada parcial de segundo orden de la Gaussiana $\frac{\partial^2}{\partial x^2} g\ (\sigma)$ con imagen *I* en el punto *p*. De manera análoga ocurre con los términos $L_{xy}(p, \sigma)$ y $L_{yy}(p, \sigma)$ de la matriz.

A pesar de que los filtros gaussianos son óptimos para el análisis del espacio-escala, se ha implementado una alternativa a los filtros gaussianos en el detector SURF debido a las limitaciones que presentan: son los llamados filtros tipo caja (de sus siglas en inglés box-filters). Estos nuevos filtros aproximan las derivadas parciales de segundo orden de las gaussianas y pueden ser evaluados de manera muy rápida usando imágenes integrales, independientemente del tamaño de éstas. Las imágenes integrales son calculadas mediante la siguiente fórmula:

$$Ii_{\sum}(x,y) = \sum_{i=1}^{i \le x} \sum_{j=1}^{j \le y} I(i,j)$$

donde (x, y) representan la posición del punto en la imagen y $Ii(x, y)$ representa la intensidad de la imagen en el punto.

Una vez la imagen integral ha sido creada, se puede calcular la suma de las intensidades de una región mediante una simple operación, como se puede observar en la Figura 2.15:

$$\sum I = Ii_D + Ii_A + Ii_B + Ii_C$$

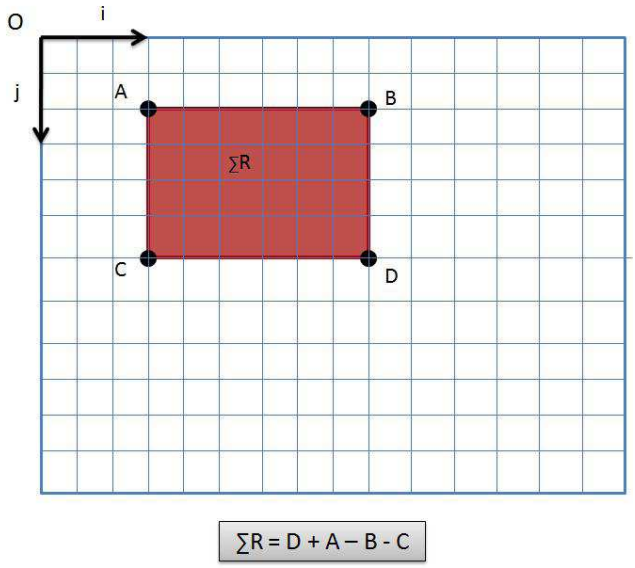


Figura 2.15: Representación de la intensidad de una región respecto de la imagen integral.

El espacio escala del descriptor SIFT descrito anteriormente, se crea a partir de imágenes suavizadas repetidamente mediante la aplicación de un filtro gaussiano y que posteriormente se submuestrean para alcanzar un nivel más alto dentro de la pirámide de dicho espacio escala. Sin embargo en el caso del detector SURF, debido a la utilización de filtros de tipo caja e imágenes integrales, no es necesario aplicar el mismo filtro iterativamente a la salida de una capa filtrada previamente, sino que se pueden aplicar dichos filtros de cualquier tamaño a la misma velocidad directamente sobre la imagen original. De este modo resulta que el espacio escala es analizado mediante la elevación del tamaño del filtro, en vez de reducir el tamaño de la imagen como es el caso del detector SIFT, como se puede observar en la Figura 2.16.

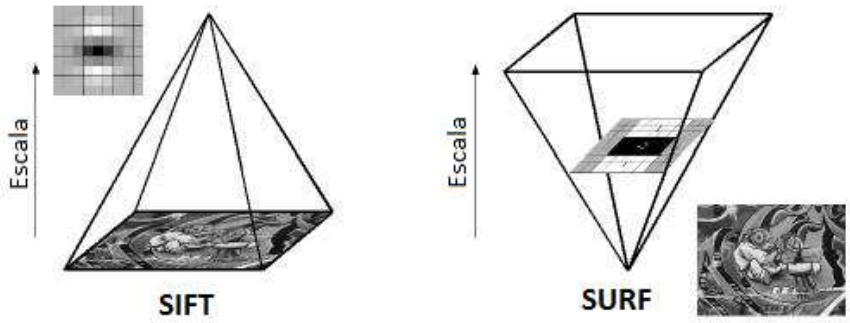


Figura 2.16: Comparación de espacios-escala entre SIFT y SURF.

Las aproximaciones de las derivadas parciales se denotan como D_{xx}, D_{xy} y D_{yy}. En cuanto al determinante de la matriz Hessiana, éste queda definido de la siguiente manera:

$$\det(H_{aprox.}) = D_{xx}D_{yy} - (0.9D_{xy})^2$$

donde el valor de 0;9 está relacionado con la aproximación del filtro gaussiano.

En la Figura 2.17 se puede observar la representación de la derivada parcial de segundo orden de un filtro gaussiano discretizado y la aproximación de la derivada implementada en el caso del descriptor SURF.

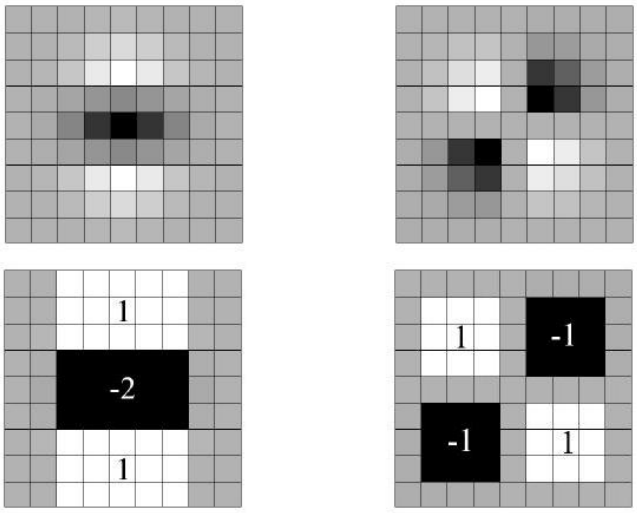


Figura 2.17: Derivadas parciales de segundo orden de un filtro gaussiano (arriba) y su aproximación (abajo).

La imagen de salida obtenida tras la convolución de la imagen original con un filtro de dimensiones 9 x 9, que corresponde a la derivada parcial de segundo orden de una gaussiana con $\sigma = 1{,}2$, es considerada como la escala inicial o también como la máxima resolución espacial (s = 1,2, correspondiente a una gaussiana con $\sigma = 1{,}2$). Las capas sucesivas se obtienen mediante la aplicación gradual de filtros de mayores dimensiones, evitando así los efectos de aliasing en la imagen.

El espacio escala para el descriptor SURF, al igual que en el caso del descriptor SIFT, está divido en octavas. Sin embargo, en el descriptor SURF, las octavas están compuestas por un número fijo de imágenes como resultado de la convolución de la misma imagen original con una serie de filtros cada vez más grandes. El incremento o paso de los filtros dentro de una misma octava es el doble respecto del paso de la octava anterior, al mismo tiempo que el primero de los filtros de cada octava es el segundo de la octava predecesora. De esta manera obtenemos las siguientes series de octavas con sus respectivos filtros como se muestra en la Figura 2.18:

- Octava inicial: $9 \times 9 \xrightarrow{6} 15 \times 15 \xrightarrow{6} 21 \times 21 \xrightarrow{6} 27 \times 27$
- Octava siguiente: $15 \times 15 \xrightarrow{12} 27 \times 27 \xrightarrow{12} 39 \times 39 \xrightarrow{12} 51 \times 51$
- Octava siguiente: $27 \times 27 \xrightarrow{24} 51 \times 51 \xrightarrow{24} 75 \times 75 \xrightarrow{24} 99 \times 99$
- Y así sucesivamente…

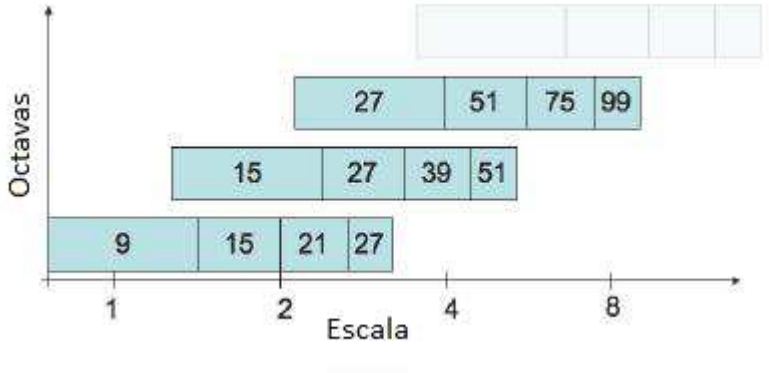


Figura 2.18: Representación gráfica de la longitud de los filtros de diferentes octavas.

Finalmente para calcular la localización de todos los puntos de interés en todas las escalas, se procede mediante la eliminación de los puntos que no cumplan la condición de máximo en un vecindario de 3 x 3 x 3. De esta manera, el máximo determinante de la matriz Hessiana es interpolado en la escala y posición de la imagen. En este punto se da por concluida la etapa de detección de los puntos de interés.

Asignación de la orientación

La siguiente etapa en la creación del descriptor corresponde a la asignación de la orientación de cada uno de los puntos de interés obtenidos en la etapa anterior. Es en esta etapa donde se otorga al descriptor de cada punto la invarianza ante la rotación mediante la orientación del mismo.

El primer paso para otorgar la mencionada orientación consiste en el cálculo de la respuesta de Haar en ambas direcciones *x* e *y* mediante las funciones representadas en la Figura 2.19.

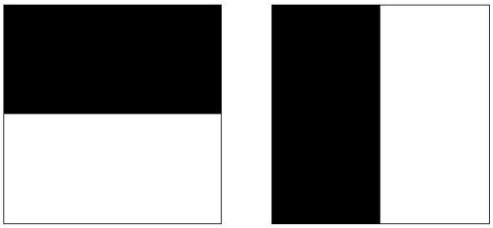


Figura 2.19: Filtros de Haar empleados en el descriptor SURF.

El área de interés para el cálculo es el área circular centrada en el punto de interés y de radio 6*s*, siendo s la escala en la que el punto de interés ha sido detectado. De la misma manera, la etapa de muestreo depende de la escala y se toma como valor *s*. Respecto de las funciones onduladas de Haar, se toma el valor 4*s*, por tanto dependiente también de la escala, como referencia, donde a mayor valor de escala mayor es la dimensión de las funciones onduladas.

Tras haber realizado todos estos cálculos, se utilizan imágenes integrales nuevamente para proceder al filtrado mediante las máscaras de Haar y obtener así las respuestas en ambas direcciones. Son necesarias únicamente 6 operaciones para obtener la respuesta en la dirección *x* e *y*. Una vez que las respuestas onduladas han sido calculadas, son ponderadas por una gaussiana de valor $\sigma = 2{,}5s$ centrada en el punto de interés. Las respuestas son representadas como vectores en el espacio colocando la respuesta horizontal y vertical en el eje de abscisas y ordenadas respectivamente. Finalmente, se obtiene una orientación dominante por cada sector mediante la suma de todas las respuestas dentro de una ventana de orientación móvil cubriendo un ángulo de $\frac{\pi}{3}$. La representación de este puede observarse en la Figura 2.20.

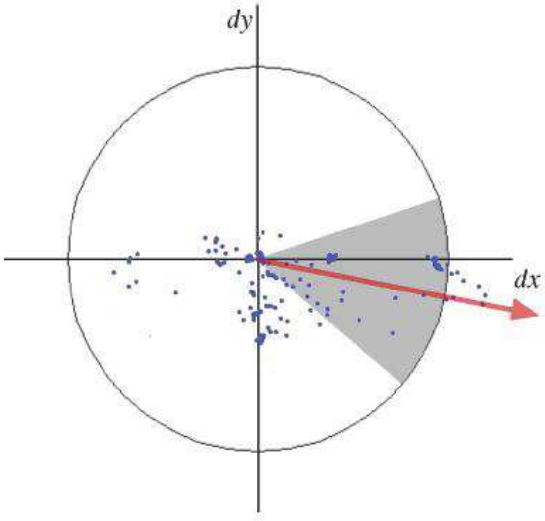


Figura 2.20: Asignación de la orientación de cada sector.

La orientación final del punto de interés será aquella cuyo vector sea el más grande dentro de los 6 sectores en los que ha sido dividida el área circular alrededor del punto de interés.

Creación del descriptor

Es en esta última etapa del proceso donde se concreta la creación del descriptor SURF.

Se construye como primer paso una región cuadrada de tamaño 20*s* alrededor del punto de interés y orientada en relación a la orientación calculada en la etapa anterior. Esta región es a su vez dividida en 4 x 4 sub-regiones dentro de cada una de las cuales se calculan las respuestas de Haar de puntos con una separación de muestreo de 5 x 5 en ambas direcciones. Por simplicidad, se consideran d_x y d_y las respuestas de Haar en las direcciones horizontal y vertical respectivamente relativas a la orientación del punto de interés. En la Figura 2.21 están representadas tanto las respuestas de Haar en cada una de las sub-regiones como las componentes d_x y d_y uno de los vectores.

Para dotar a las respuestas d_x y d_y de una mayor robustez ante deformaciones geométricas y errores de posición, éstas son ponderadas por una gaussiana de valor $\sigma = 3{,}3s$ centrada en el punto de interés.

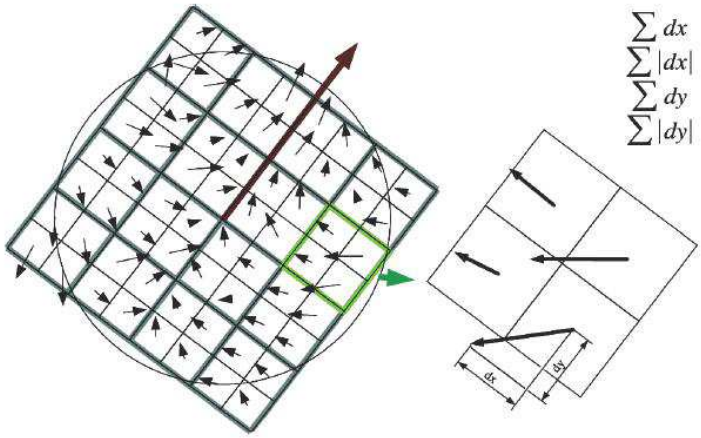


Figura 2.21: Respuestas de Haar en las sub-regiones alrededor del punto de interés.

En cada una de las sub-regiones se suman las respuestas d_x y d_y obteniendo así un valor de dx y dy representativo por cada una de las sub-regiones. Al mismo tiempo se realiza la suma de los valores absolutos de las respuestas $|d_x|$ y $|d_y|$ en cada una de las sub-regiones, obteniendo de esta manera, información de la polaridad sobre los cambios de intensidad.

En resumen, cada una de las sub-regiones queda representada por un vector *v* de componentes:

$$v = \left(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|\right)$$

y por lo tanto, englobando las 4 x 4 sub-regiones, resulta un descriptor SURF con una longitud de 64 valores para cada uno de los puntos de interés identificados.

Figura 2.22.- Ejemplo de puntos clave SURF detectados en una imagen.

BIBLIOGRAFÍA DEL CAPÍTULO 2.1

[Esc01] [Low99] [Bay+06] [SURF]

2.2 LIBRERÍAS DE VISIÓN POR COMPUTADOR

En el mercado actual existen una gran cantidad de librerías programadas en varios lenguajes que nos proporcionan las herramientas necesarias para crear aplicaciones de visión por computador.

Este apartado contiene una pequeña introducción a algunas de las más usadas. Son OpenCV, IVT, Javavis y el Image Processing Toolbox de MATLAB.

2.2.1 LIBRERÍA OPENCV

OpenCV (Open Source Computer Vision) es una librería de funciones de programación dirigida principalmente a la visión por ordenador en tiempo real, desarrollada por Intel en 1999 y ahora con el apoyo de Willow Garage.

Esta librería se centra principalmente en el procesamiento de imágenes en tiempo real. La biblioteca es multiplataforma y una de sus principales ventajas es que puede ejecutarse en ordenadores sin grandes prestaciones, pudiendo realizarse aplicaciones de visión bastante sofisticadas y a buena velocidad.

Figura 2.23: Logotipo de Willow Garage

La biblioteca fue originalmente escrita en C, y esta interfaz de C hace OpenCV portátil para algunas plataformas específicas, tales como procesadores de señal digital. Contenedores para lenguajes como C++, Python, Ruby y Java (usando JavaCV) se han desarrollado para fomentar la adopción a un público más amplio.

Desde la versión 2.0 OpenCV incluye tanto su interfaz de C tradicional, así como una nueva interfaz C++. Esta nueva interfaz busca reducir el número de líneas de código necesario para codificar la funcionalidad de la visión, así como de reducir los errores de programación comunes, tales como pérdidas de memoria (a través de la asignación automática de datos y cancelación de asignación) que pueden surgir al utilizar OpenCV en C.

Desde que se desarrolló el número de aplicaciones se va incrementando diariamente, algunas de estas aplicaciones pueden ser:

- Interacción entre computadores y personas (HCI)
- Identificación de objetos, segmentación y reconocimiento
- Reconocimiento de gestos faciales
- Estructuras de movimiento (SFM)
- Aplicaciones en robots móviles
- Control de procesos
- Reconstrucción 3D y calibración de la cámara
- Estabilización de imágenes

Una de las principales ventajas de OpenCV es su versatilidad y fácil portabilidad de unos sistemas a otros, además de que posee licencia de código abierto BSD y su uso es gratuito. Es compatible con un gran número de sistemas operativos, entre los que se encuentran Windows, MacOS, Linux, FreeBSD, Maemo, OpenBSD, e incluso algunos portátiles como Android o iOS.

2.2.2 LIBRERÍA IVT

Integrating Vision Toolkit (IVT) es una potente y rápida librería de C++ dedicada a la visión por computador, con una arquitectura fácil de usar y orientada a objetos. Fue desarrollada por la anteriormente conocida como Universidad de Karlsruhe, ahora conocida como "Karlsruhe Institute of Technology" (KIT).

Desde 2009, IVT es mantenida en cooperación con la compañía Keyetech, que también ofrece cursos de formación para el uso de IVT así como de sus productos comerciales y soluciones a medida usando el IVT.

Figura 2.24: Logotipo del Karlsruhe Institute of Technology

El paquete de software que incluye clases para la captura, procesamiento, presentación, sincronización, creación de hilos y algoritmos de procesamiento. Tiene una interfaz genérica para cada tipo de componente que permite el uso de distintos interfaces sobre Windows, Linux y MacOS con una interfaz común. Los formatos internos de imagen son RGB y escala de grises.

2.2.3 LIBRERÍA AFORGE.NET

AForge.NET es una herramienta de visión por computador e inteligencia artificial diseñada para desarrolladores e investigadores, compuesta por varias librerías dependiendo de la aplicación deseada:

- AForge.Imaging: librerías de procesado de imágenes.
- AForge.Vision: librerías de detección de movimiento.
- AForge.Math: librerías de algoritmos matemáticos.
- AForge.Video: librerías de tratamiento de vídeos.
- Otras librerías para usos más específicos como el control de robots o arquitecturas neuronales.

Figura 2.25: Logotipo de AForge.NET

La librería se mejora continuamente con el lanzamiento de nuevas versiones y posee una buena documentación en diferentes librerías y fuentes. Está programada en C++ y únicamente es compatible con el sistema operativo de Microsoft, Windows.

2.2.4 LIBRERÍA JAVAVIS

JavaVis es una librería de visión artificial, escrita íntegramente en Java. Está basada en la librería Vista, una librería de visión que se desarrolló en C y que pretendía simular la programación orientada a objetos.

Figura 2.26: Logotipo de Java

La librería se basa en unas clases básicas, que sirven para poder desarrollar métodos más complejos. Las principales ventajas que presenta son:

- Adaptada para plataformas portátiles.
- Posibilidad de ampliación fácilmente.
- Una interfaz gráfica.
- Posee un formato de imagen especial que permite almacenar secuencias de imágenes y datos de tipo geométrico.

Actualmente JavaVis se encuentra en fase de desarrollo de una nueva versión cuyo lanzamiento está previsto para finales de 2012. Se trata de una librería de uso libre y está disponible para los sistemas operativos Windows, Linux y FreeBSD.

2.2.5 LIBRERÍA IMAGE PROCESSING TOOLBOX

Image Processing Toolbox es una herramienta desarrollada para el software de cálculo matemático MATLAB, proporcionándole un conjunto de funciones que amplía sus capacidades para realizar el desarrollo de aplicaciones y de nuevos algoritmos en el campo del proceso y análisis de imágenes. El entorno matemático de MATLAB es ideal para el procesado de imágenes, ya que estas imágenes son, al fin y al cabo, matrices. Está disponible para Windows, Linux, MacOS y Solaris.

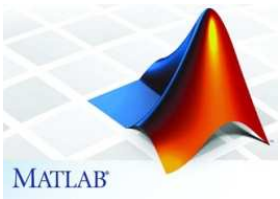


Figura 2.27: Logotipo de MATLAB

Algunas de las funciones más importantes que incluye son:

- Análisis de imágenes y estadística.
- Diseño de filtros y recuperación de imágenes.
- Mejora de imágenes.
- Operaciones morfológicas y geométricas.
- Definición de mapas de colores y modificación gráfica.
- Transformación de imágenes.

2.2.6 OTRAS LIBRERÍAS

Además de las mencionadas existen multitud de librerías que poseen una menor relevancia en cuanto a la extensión de su uso se refiere. Algunas de ellas son:

- SIP (Scilab Image Processing)
- MSFM (Modular Flow Scheduling Middleware Framework)
- 3DSlicer
- ANIMAL
- SigmaScan
- VXL

Figura 2.28: Logotipos de diversas bibliotecas de visión por computador

BIBLIOGRAFÍA DEL CAPÍTULO 2.2

[Bra+08] [OPENCV] [AFORGE] [IVT] [IPT] [JVis] [MFSM]

CAPÍTULO 3

HERRAMIENTAS Y PLATAFORMA UTILIZADA

3.1. HARDWARE

Este capítulo recoge las características técnicas de los principales recursos hardware utilizados para la realización del proyecto. Éstos consisten básicamente en un ordenador portátil para el procesamiento de las imágenes y una cámara web para la captura de éstas.

3.1.1. HARDWARE DE PROCESAMIENTO DE IMÁGENES

Para la realización del proyecto se ha utilizado un ordenador portátil Sony Vaio VPCEA3S1E con las siguientes características:

- Procesador Intel Core i3 M370 2.40Ghz.
- Tarjeta Gráfica ATI Radeon HD5470 512MB GDDR5.
- 4 GB de Memoria RAM DDR3-SDRAM.

Figura 3.1.- Sony Vaio VPCEA3S1E

3.1.2. HARDWARE DE CAPTURA DE IMÁGENES

Pese a que el ordenador portátil cuenta con cámara web integrada, hemos contado con una cámara con conexión USB para facilitar y simular el movimiento que realizaría el robot. En concreto se trata de una Logitech Quickcam Pro 9000, con las siguientes características técnicas:

- Resolución máxima: 2 megapíxeles.
- Óptica Carl Zeiss.
- Enfoque automático.
- Conexión USB 2.0.

Figura 3.2.- Logitech Quickcam Pro 9000

BIBLIOGRAFÍA DEL CAPÍTULO 3.1

[SONY] [LOGI]

3.2. SOFTWARE

En este capítulo se describen los principales recursos software utilizados, como el sistema operativo Ubuntu, la librería OpenCV 2.1 y el software de edición fotográfica GIMP incluido en el sistema operativo.

3.2.1. SISTEMA OPERATIVO UBUNTU

Ubuntu es un sistema operativo que utiliza núcleo Linux, basado en el código base del proyecto Debian. El principal motivo para la elección de este operativo es que proporciona una gran facilidad y fiabilidad a la hora de programar en C y C++. Está mantenido y desarrollado por Canonical, una compañía británica, y por la comunidad de desarrolladores.

Figura 3.3: Logotipo de Ubuntu

El sistema se ofrece de forma gratuita gracias a que usa primariamente software libre, haciendo excepciones para varios controladores privativos y firmware y software no libre incluido en el kernel Linux y en sus repositorios.

Otra ventaja que posee es la amplia comunidad de desarrolladores que se encargan de mantener el sistema constantemente actualizado.

Cada seis meses se publica una nueva versión de Ubuntu la cual recibe soporte por parte de Canonical, durante dieciocho meses. Durante la realización del proyecto el sistema operativo ha sido actualizado en varias ocasiones, y las versiones usadas han sido la 11.04 *"Natty Narwhal"* y la 11.10 "*Oneiric Ocelot".*

3.2.2 LENGUAJE DE PROGRAMACIÓN C

El lenguaje de programación escogido es C. Fue inventado e implementado por primera vez por Dennis M. Ritchie en los laboratorios Bell como resultado de un proceso de desarrollo comenzado con un lenguaje anterior denominado B, inventado por Kenneth L. Thompson. En 1978, Brian Kernighan y Dennis Ritchie publicaron el libro The C Programming Language que ha servido hasta la actualidad como definición eficiente de este lenguaje.

Figura 3.4: Kenneth L. Thompson (izquierda) y Dennis M. Ritchie (derecha), creadores del lenguaje C.

Durante muchos años el estándar de C fue la versión proporcionada con la versión cinco del sistema operativo UNIX. En 1983, el instituto de estándares americanos estableció un estándar que definiera el lenguaje C, conocido como ANSI C. Hoy día, todos los principales compiladores de C llevan implementado el estándar ANSI.

El lenguaje C se denomina como un lenguaje de nivel medio, puesto que combina elementos de lenguajes de alto nivel con el funcionalismo del lenguaje ensamblador.

3.2.2.1 CARACTERÍSTICAS

Pese a la antigüedad del lenguaje todavía se sigue utilizando en muchos desarrollos de aplicaciones. Las principales características que presenta son:

- Permite la manipulación de elementos básicos de computación como bits, bytes y direcciones.
- Un núcleo del lenguaje simple, con funcionalidades añadidas importantes, como funciones matemáticas y de manejo de ficheros, proporcionadas por bibliotecas.
- Una gran flexibilidad que permite programar con múltiples estilos.
- Posee un sistema de tipos que impide operaciones sin sentido.
- Usa un lenguaje de pre-procesado para tareas como definir macros e incluir múltiples ficheros de código fuente.
- Fácil acceso a memoria de bajo nivel mediante el uso de punteros.
- Interrupciones al procesador con uniones.
- Un conjunto reducido de palabras clave, solamente 32 (27 definidas en la versión original y 5 añadidas por el comité ANSI)
- Por defecto, el paso de parámetros a una función se realiza por valor. El paso por referencia se consigue pasando explícitamente a las funciones las direcciones de memoria de dichos parámetros.
- Punteros a funciones y variables estáticas, que permiten una forma rudimentaria de encapsulado y polimorfismo.

Pese a que nuestro programa está escrito en C, incluye una función proporcionada por OpenCV que está escrita en C++, destinada a la detección de puntos clave mediante el uso de la librería FLANN. Por este motivo se han usado archivos con la extensión *.cpp* y el compilador *gcc,* propios de C++.

3.2.3 LIBRERIA OPENCV 2.1

Anteriormente ya hemos hablado de la historia de esta librería y de sus principales características, por lo que en este capítulo abarcaremos los principales tipos de datos y distintas funciones de OpenCV que hemos utilizado para elaborar nuestro programa.

Figura 3.5: Logotipo de OpenCV

Cabe destacar que hemos utilizado la versión 2.1 de OpenCV, lanzada en abril de 2010. En la actualidad la versión más reciente es la 2.3.1, lanzada en agosto de 2011.

3.2.3.1 TIPOS DE DATOS UTILIZADOS

La librería OpenCV dispone de múltiples tipos de datos específicos para desenvolverse con facilidad en cada área de la visión por computador. En este apartado únicamente abordaremos los utilizados por nuestro programa.

CvPoint

Define las coordenadas de un punto. Posee dos campos, el valor entero de la coordenada *X* y el de *Y.*

CvPointD32f

También define las coordenadas de un punto pero en este caso en vez de con enteros utiliza el tipo de dato *float*, número en coma flotante.

CvRect

Define la posición y tamaño de un rectángulo. Posee cuatro campos, dos para las posciones *X* e *Y* de la esquina superior izquierda del rectángulo, y otros dos con el ancho y largo de éste. La definición del constructor es la siguiente:

```
CvRect cvRect (int x, int y, int width, int height);
```

CvScalar

La estructura *CvScalar* es un contenedor de hasta 4 valores de tipo *double.* Útil para almacenar el valor de un píxel en los diferentes canales.

IplImage

Formato de imagen de OpenCV. Las imágenes están compuestas por una cabecera y una zona de datos. La estructura está formada por campos relativos a la imagen, de los cuales sólo los más importantes describimos a continuación:

- *Width* y *height:* Ancho y alto de la imagen.
- *Depth*: Tipo de valor que puede tomar cada píxel. Las opciones disponibles son:
  IPL_DEPTH_8U Unsigned 8-bit integer (8u)
  IPL_DEPTH_8S Signed 8-bit integer (8s)
  IPL_DEPTH_16S Signed 16-bit integer (16s)
  IPL_DEPTH_32S Signed 32-bit integer (32s)
  IPL_DEPTH_32F 32-bit floating-point single-precision (32f)
  IPL_DEPTH_64F 64-bit floating-point double-precision (64f)
- *NChanels:* Número de planos que tiene la imagen, 1 para imágenes monocromo y 3 para imágenes a color.
- *Origin:* Origen de los ejes de coordenadas de la imagen.
- *ImageData*: Puntero a la primera final de la imagen.
- *Roi:* Estructura que nos permite operar solo sobre una región de la imagen (región de interés), sobre un solo plano o ambas cosas a la vez. La estructura Roi está compuesta por cinco enteros que indican los planos, además del origen y tamaño de la región de interés.

CvCapture

La estructura de captura de vídeo. No posee interface pública y es usada solamente como parámetro para las operaciones de captura de vídeo.

CvSize

Estructura para definir el tamaño de una imagen. Posee dos campos:

- *Width:* Ancho de la imagen.
- *Height:* Alto de la imagen.

CvMemStorage

Estructura de los almacenes de memoria, que proporcionan los bloques de memoria necesarios para guardar las demás estructuras. Están compuestos por una cabecera seguida de una lista de direcciones y cantidades de memoria disponible.

CvSeq

Una secuencia es un array de tamaño variable de diferentes tipos de elementos almacenado en un almacén de memoria. La secuencia es dividida en partes y cada una se almacena en un bloque de memoria. Cada uno, apunta al siguiente y al anterior.

CvMat

Estructura que define a las matrices. Puede contener números reales de precisión simple o doble. Hay funciones que permiten hacer operaciones aritméticas sencillas y otras más complejas que pueden calcular los autovalores o la descomposición en valores singulares.

CvSURFParams

Estructura que almacena los parámetros del algoritmo SURF necesarios para extraer los puntos clave y descriptores de una imagen. El constructor es:

```
CvSURFParams cvSURFParams(double hessianThreshold, int
extended=0);
```

3.2.3.2 FUNCIONES UTILIZADAS

En este apartado se definirán algunas de las funciones utilizadas y se explicarán brevemente los parámetros de cada una:

cvCreateImage

Inicializa una imagen, crea la cabecera y reserva memoria. Devuelve un puntero a la cabecera de la imagen creada.

```
IplImage* cvCreateImage( CvSize size, int depth, int channel);
```

- *Size*: tamaño de la imagen.
- *Depth*: tipo de datos para los píxeles
- *Channels*: número de canales

cvReleaseImage

Libera la cabecera y memoria de una imagen.

```
void cvReleaseImage( IplImage** image );
```

- *Image*: doble puntero a la cabecera de la imagen.

cvReleaseImageHeader

Libera únicamente la cabecera de una imagen.

```
void cvReleaseImageHeader( IplImage** image );
```

- *Image*: doble puntero a la cabecera de la imagen.

cvSetImageROI

Establece la región de interés de una imagen, delimitada por el rectángulo dado.

```
void cvSetImageROI (IplImage* image, CvRect rect);
```

- *Image*: puntero a la imagen.
- *Rect*: rectángulo que determina la región de interés.

cvResetImageROI

Elimina el área de interés creado para incluir la imagen completa y libera la estructura ROI.

```
void cvResetImageROI (IplImage* image);
```

cvCopyImage

Copia una imagen en otra. Las imágenes o las regiones de interés que se copiarán deben ser del mismo tamaño.

```
void cvCopyImage(IplImage* src, IplImage* dst);
```

- *Src*: puntero a la imagen de referencia.
- *Dst*: puntero a la imagen de destino.
- *Img*: puntero a la imagen.

cvNamedWindow

Crea una ventana para visualizar imágenes.

```
int cvNamedWindow(const char* name, int flags);
```

- *Name*: nombre de la ventana.
- *Flags*: opciones de la ventana.

cvShowImage

Dibuja una imagen en la ventana especificada.

```
void cvShowImage(const char* name, const CvArr* image);
```

- *Name*: nombre de la ventana.
- *Image*: puntero a la imagen que mostrar.

cvDestroyWindow

Destruye una ventana.

```
void cvDestroyWindow(const char* name);
```

- *Name*: nombre de la ventana.

cvCvtColor

Convierte la imagen de un espacio de color a otro.

```
void cvCvtColor (const CvArr* src, CvArr* dst, int code);
```

- *Src*: puntero a la imagen de referencia.
- *Dst*: puntero a la imagen de destino.
- *Code*: operación de conversión de color. Especificada de la forma *CV_*src_color_space*2*dst_color_space*.*

cvCreateMat

Crea la cabecera y almacena los datos de una matriz.

```
CvMat* cvCreateMat (int rows, int cols, int type);
```

- *Rows*: Número de filas de la matriz.
- *Cols*: Número de columnas de la matriz.
- *Type*: Tipo de los elementos de la matriz. Definido como *CV_<bit depth><S|U|F><number of channels>.*

cvWaitKey

Espera la pulsación de una tecla.

```
int cvWaitKey (int delay=0);
```

- *Delay*: retraso en milisegundos.

cvGet2D

Devuelve el valor específico de un elemento de un array.

```
CvScalar cvGet2D (const CvArr* arr, int idx0, int idx1);
```

- *Arr*: array de origen.
- *Idx0, idx1*: índices del elemento.

cvSet2D

Cambia el valor de un elemento de un array.

```
CvScalar cvSet2D (const CvArr* arr, int idx0, int idx1);
```

- *Arr*: array de origen.
- *Idx0, idx1*: índices del elemento.

cvExtractSURF

Extrae las características SURF de una imagen.

```
void cvExtractSURF(const CvArr* image, const CvArr* mask,
CvSeq** keypoints, CvSeq** descriptors, CvMemStorage* storage,
CvSURFParams params);
```

- *Image*: la imagen de entrada. Debe ser de 8 bits monocroma.
- *Mask*: máscara de 8 bits opcional.
- *Keypoints*: parámetro de salida, doble puntero a la secuencia de puntos clave.
- *Descriptors*: parámetro de salida opcional, doble puntero a la secuencia de descriptores.
- *Storage*: almacén de memoria donde los puntos clave y descriptores serán guardados.
- *Params*: varios parámetros del algoritmo definidos por la estructura *CvSURFParams.*

cvGetPerspectiveTransform

Calcula la transformación en perspectiva a partir de 4 puntos.

```
CvMat* cvGetPerspectiveTransform(const CvPoint2D32f* src, const
CvPoint2D32f* dst, CvMat* mapMatrix);
```

- *Src*: coordenadas de los 4 vértices de la imagen de origen.
- *Dst*: coordenadas de los 4 vértices correspondientes de la imagen de destino.
- *MapMatrix*: Puntero a la matriz 3x3 de destino.

cvWarpPerspective

Aplica la transformación en perspectiva a una imagen.

```
void cvWarpPerspective(const CvArr* src, CvArr* dst, const
    CvMat* mapMatrix, int flags, CvScalar fillval);
```

- *Src*: imagen de origen.
- *Dst*: imagen de destino.
- *MapMatrix*: matriz de transformación 3x3.
- *Flags*: métodos de interpolación.
- *Fillval*: valor que se usa para rellenar los valores exteriores.

cvCaptureFromCAM

Inicializa la captura de vídeo de la cámara.

```
CvCapture* cvCaptureFromCAM (int index);
```

- *Index*: índice de la cámara que se desea utilizar. Si únicamente hay una cámara o no importa qué cámara usar se puede indicar con el índice -1.

cvQueryFrame

Devuelve un puntero a una captura de un frame de la cámara.

```
IplImage* cvQueryFrame (CvCapture* capture);
```

- *Capture*: estructura de captura de vídeo.

cvReleaseCapture

Libera la estructura *CvCapture.*

```
void cvReleaseCapture (CvCapture** capture);
```

- *Capture*: puntero a la estructura de captura de vídeo.

3.2.4. SOFTWARE DE EDICIÓN FOTOGRÁFICA GIMP

Para la edición de fotografías se ha utilizado el software GIMP (GNU Image Manipulation Program). Se trata de un programa gratuito de licencia libre GNU, de apariencia bastante similar al conocido Photoshop, y que viene preinstalado por defecto con el sistema operativo Ubuntu.

Figura 3.6.- Logotipo de Gimp

GIMP permite trabajar con capas, para poder modificar cada objeto de la imagen en forma totalmente independiente a las demás capas en la imagen y es compatible con la mayoría de los formatos de ficheros gráficos, entre ellos; JPG, GIF, PNG, PCX, TIFF y PSD, además de poseer su propio formato de almacenamiento de ficheros, XCF.

GIMP cuenta con numerosas herramientas, entre las que encontramos las siguientes:

- Herramientas de selección (rectangular, esférica, lazo manual, varita mágica, por color),
- Tijeras inteligentes,
- Herramientas de pintado como pincel, brocha, aerógrafo, relleno, texturas, etc.
- Herramientas de modificación de escala, de inclinación, de deformación, clonado en perspectiva o brocha de curado (para corregir pequeños defectos).
- Herramientas de manipulación de texto.
- Posee también muchas herramientas o filtros para la manipulación de los colores y el aspecto de las imágenes, como enfoque y desenfoque, eliminación o adición de manchas, sombras, mapeado de colores, etc.
- También posee un menú con un catálogo de efectos y tratamientos de las imágenes.

BIBLIOGRAFÍA DEL CAPÍTULO 3.2

[Bra+08] [Ker+78] [OPENCV] [UBUNTU] [GIMP]

CAPÍTULO 4

METODOLOGÍA EMPLEADA

4.1. FUNCIONAMIENTO DEL PROGRAMA

Este capítulo está dedicado a explicar el funcionamiento general del programa, exponiendo cada una de las fases de las que está formado, desde el momento en que se obtiene una imagen de la cámara web hasta que conseguimos la imagen panorámica de salida.

4.1.1 VISIÓN GENERAL DEL PROGRAMA

En términos generales el programa se compone de cuatro fases a la hora de realizar su labor como se representa en el diagrama de flujo de la Figura 4.1.

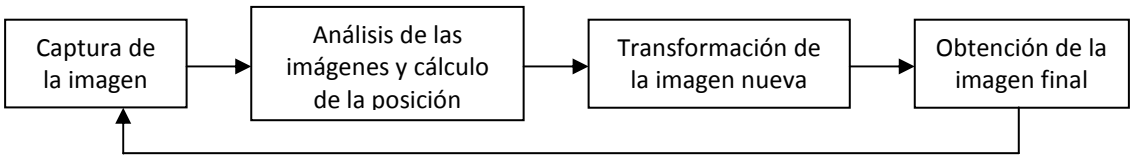


Figura 4.1: Diagrama de flujo general del programa.

La primera fase consiste principalmente en la captura de la imagen con la cámara web. Para ello, debemos comprobar que la cámara está presente y lista para capturar imágenes. En caso de no ser así, el programa se detendrá automáticamente presentando un mensaje de error. Además hemos de diferenciar la primera instantánea tomada de las demás, ya que esta primera se mostrará directamente debido a que no tenemos una imagen anterior con la que compararla. Las sucesivas se irán comparando con la imagen total anterior calculada en el anterior ciclo de funcionamiento.

Una segunda se encarga extraer las características SURF de ambas imágenes para después poder cotejarlas. Hay que tener en cuenta que el algoritmo exige que la imagen de entrada sea en escala de grises. También se calcula la posición de la nueva imagen a partir de los datos SURF obtenidos. Se comparan los puntos clave para hallar parejas, y posteriormente se calcula la posición relativa entre cada uno de ellos para obtener la posición de los vértices que deberá ocupar la nueva imagen.

La tercera etapa del proceso transforma la nueva imagen a la posición obtenida anteriormente usando una matriz de transformación. En el caso de obtener valores negativos en los vértices se corrige desplazando la imagen para trabajar siempre en el semiplano positivo.

En la última etapa se fusionan las imágenes teniendo en cuenta la posición obtenida, presentándola por pantalla. Así mismo se libera la memoria reservada para el funcionamiento del programa.

Conociendo el funcionamiento general del programa pasaremos a explicar cada una de las fases con más detenimiento.

4.1.2 PRIMERA FASE: CAPTURA DE LA IMAGEN

El primer paso es obtener la imagen de la cámara web con la función *cvQueryFrame*. La primera imagen capturada se muestra directamente y se utiliza como base sobre la que posteriormente se van colocando las imágenes que la cámara va tomando, comparando la nueva imagen con la ya existente.

La imagen que captura la cámara se muestra a tiempo real en la ventana llamada "*Webcam*", y el programa se queda a la espera de que se pulse una tecla. Las posibles teclas que podemos pulsar son dos, la tecla *ENTER* que mandará la orden de tomar la captura, y la tecla *ESC* para salir del programa. El algoritmo ignorará cualquier otra tecla pulsada.

Si el programa no detecta la señal proveniente de la cámara web o bien ésta se ha seleccionado incorrectamente se mostrará el mensaje de error *"NO SE PUEDE ACCEDER A LA WEBCAM"* y el programa finalizará.

4.1.3 SEGUNDA FASE: ANÁLISIS DE LA IMAGEN

Una vez tomada la primera imagen base y una segunda imagen para poder compararla con ésta, nuestra tarea consiste en analizar ambas imágenes. Se obtienen sus puntos característicos y descriptores, para posteriormente poder compararlos y establecer la posición que ocupa la nueva imagen respecto a la anterior.

4.1.3.1 EXTRACCIÓN DE PUNTOS CARACTERÍSTICOS

Para la detección de puntos clave y descriptores utilizamos la función *cvExtractSURF*. Esta función nos devuelve la localización, tamaño, orientación y, opcionalmente, el descriptor, que puede ser básico o extendido, de los puntos clave de una imagen.

Cuantos más objetos aparezcan en la imagen y más formas contengan éstos, más puntos clave detecta la función. Cuando la imagen no contiene demasiados objetos o relieves, sobre todo en superficies planas y lisas como paredes, suelo, superficies de color uniforme… el número de puntos clave encontrados desciende hasta cifras con las que es prácticamente imposible conseguir una comparación fiable.

De la misma manera debemos procurar que la imagen sea lo más nítida posible, ya que imágenes movidas o desenfocadas dificultan el reconocimiento de los puntos clave por parte de la función. Como problema añadido, aunque la posición de la imagen se calcule correctamente y se fusione con éxito, estos defectos provocan dificultades para realizar los cálculos de las siguientes imágenes.

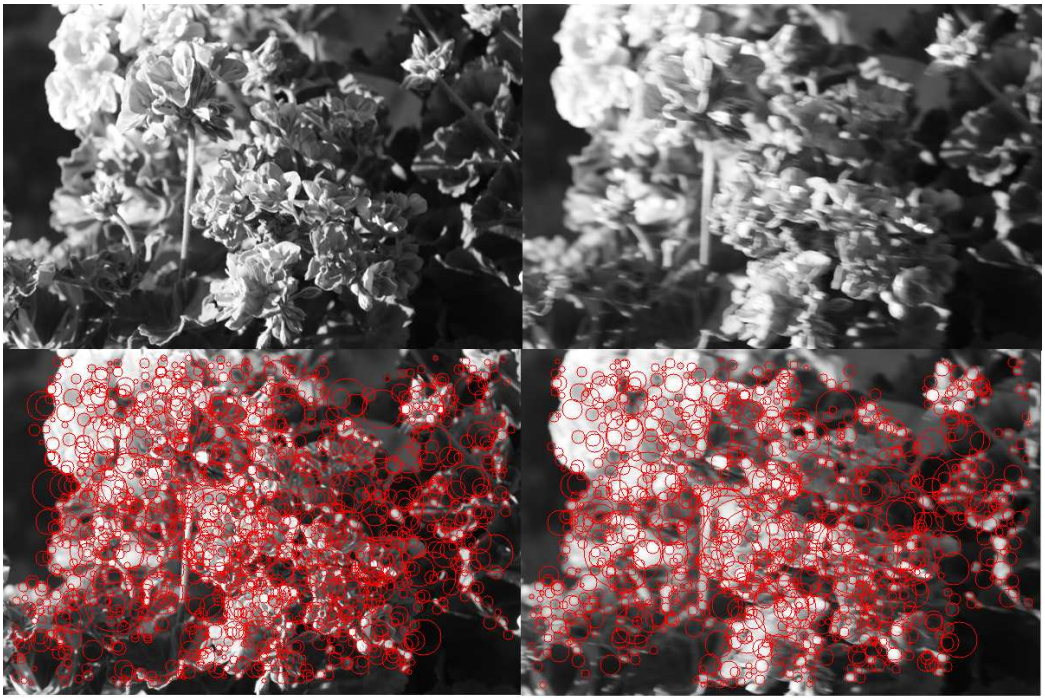


Figura 4.2: Comparación de puntos clave en imágenes movidas. a) Imagen estática con 1.549 puntos clave detectados. b) Imagen movida con 1.134 puntos clave detectados (28% menos).

4.1.3.2 CÁLCULO DE LA POSICIÓN

Una vez tenemos los puntos clave de la imagen actual y de la imagen anterior, procedemos a calcular la posición relativa entre ellas. Para ello utilizamos la función *locatePlanarObject*, proporcionada en la documentación y ejemplos de la propia librería OpenCV, y cuyo cometido es utilizar los puntos clave y descriptores de cada imagen para intentar conseguir una correspondencia entre ambas imágenes, calculando parejas que cumplan las mismas características.

```
int locatePlanarObject(const CvSeq* objectKeypoints, const
CvSeq* objectDescriptors, const CvSeq* imageKeypoints, const
CvSeq* imageDescriptors,const CvPoint src_corners[4], CvPoint
dst_corners[4], const CvPoint res[4])
```

La búsqueda de los pares se puede realizar mediante dos procedimientos dependiendo del uso o no de la librería FLANN (Fast Library for Approximate Nearest Neighbors). Esta librería contiene una serie de algoritmos optimizados para la rápida búsqueda de pares en imágenes de gran tamaño.

En el programa se incluyen las funciones necesarias para realizar la búsqueda mediante ambos procedimientos. Nosotros por defecto realizamos la búsqueda mediante la librería FLANN, pese a que en las numerosas pruebas experimentales realizadas con ambos métodos no concluyen una gran diferencia ni de velocidad ni de precisión entre ellos.

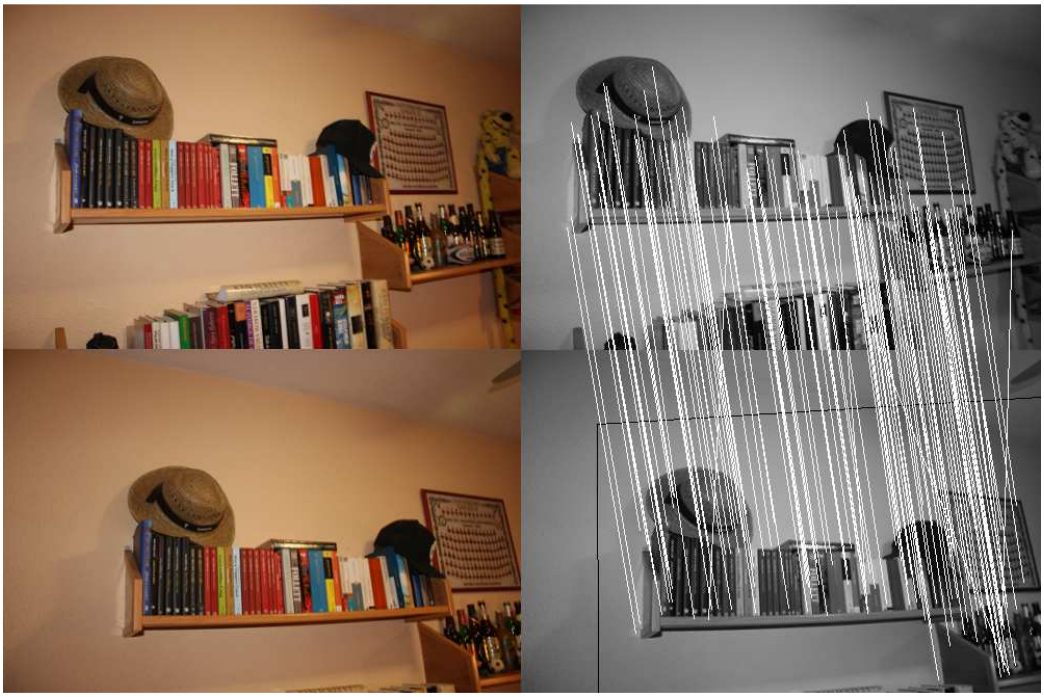


Figura 4.3: Localización de los pares de puntos clave y posición relativa de las imágenes.

En la Figura 4.3 se muestra un ejemplo de búsqueda de parejas de puntos clave entre dos imágenes. Este número de pares que presentan correspondencia ha de ser mayor que un número determinado para asegurar un correcto acoplamiento de las instantáneas.

Al término de las diversas pruebas que hemos realizado, estimamos que un número óptimo a partir del cual se pueden obtener buenos resultados en la mayoría de los casos es el de 15 parejas de puntos clave, fijándolo así en el programa. Por debajo de ese número de pares la posición se calcula erróneamente en un porcentaje demasiado elevado de veces.

La función *locatePlanarObject* además de calcular las parejas de puntos clave, calcula la posición relativa de cada una de ellas, es decir evalúa la imagen delimitada por los vértices de origen *src*, y calcula la posición de la imagen nueva guardando sus vértices en *dst*, representados en la Figura 4.4.

En el caso de que la función no obtuviera suficientes parejas de puntos clave entre las dos imágenes se cancela el resto del proceso. Por lo tanto no se realiza el acoplamiento entre ellas, no se guarda ningún dato obtenido y el programa regresa al punto de espera de entrada de una nueva imagen.

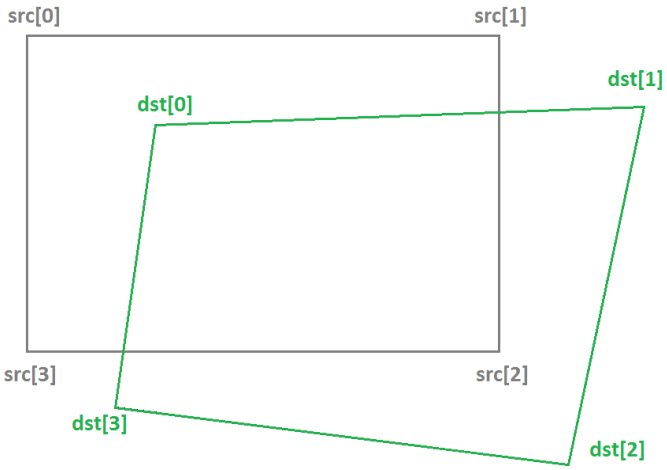


Figura 4.4: Vértices de origen *src* y de destino *dst.*

4.1.3.3 OTROS CÁLCULOS

Con los vértices de origen y los datos obtenidos acerca de los vértices de destino se calculan otra serie de variables importantes para los cálculos posteriores. Se evalúan los vértices y se calculan los valores máximos y mínimos que se alcanzan en ambos ejes de coordenadas. También se guardan datos acerca del tamaño de ambas imágenes.

4.1.4 TERCERA FASE: TRANSFORMACIÓN DE LA IMAGEN

Una vez conocemos la posición en la que debe situarse la nueva imagen para un perfecto fusionado, procedemos a transformarla para que se acople debidamente. Debemos tener en cuenta que si la cámara se mueve hacia la izquierda y/o hacia arriba, obtendremos valores negativos en los vértices de destino *dst*, por lo que será necesario trasladar el resto de la imagen antes de fusionarlas.

4.1.4.1 ENCUADRE DE LA IMAGEN

A partir de los máximos y mínimos calculados anteriormente se crea un rectángulo delimitado por los vértices llamados *esq[4]*, del tamaño exacto de la futura imagen fusionada. La función encargada de esta tarea es la llamada *encuadrar*.

```
void encuadrar(const int minX, const int maxX, const int minY,
  const int maxY, const CvPoint src_corners[4], const CvPoint
                dst_corners[4], CvPoint   esq[4])
```

Estos vértices pueden ser corregidos posteriormente si contienen datos negativos y serán los vértices de origen para la siguiente transformación.

4.1.4.2 TRANSFORMACIÓN DE LA IMAGEN

El siguiente paso es la transformación de la imagen siguiendo la posición de los vértices de destino calculados. Para ello utilizamos la función bautizada *transformar*.

```
IplImage* transformar(const CvSize tam, const CvPoint2D32f
origen[4], const CvPoint2D32f destino[4], constIplImage* im1)
```

Esta función realiza una transformación en perspectiva de la imagen extraída de la cámara y la sitúa exactamente en la posición delimitada por los vértices de destino, como podemos observar en la Figura 4.5.

a) b)

Figura 4.5: Ejemplo de transformación perspectiva de la imagen.
a) Imagen sin transformar. b) Imagen después de la transformación.

Si debido al movimiento de la cámara alguno de éstos se encuentra en el semiplano negativo, toda la información que contenga esa parte de la imagen que se encuentra en la zona de valores negativos se perderá, ya que la ventana de presentación no es capaz de mostrar valores negativos, como se observa en la Figura 4.6.

a) b)

Figura 4.6: Ejemplo de imagen recortada.
a) Imagen resultante sin realizar el rectificado. La ventana de presentación no es capaz de mostrar valores negativos. b) Imagen resultante tras el rectificado.

Si se nos presenta este caso debemos rectificar previamente las posiciones, tanto de la imagen anterior como las posiciones de destino de la nueva, para asegurarnos trabajar en el semiplano positivo. Esta tarea es llevada a cabo por la función *rectificado.*

```
IplImage* rectificado(IplImage* cor1, const CvSize tam, const
CvPoint dst_corners[4], CvPoint esq[4], const int minX, const
                int minY, CvPoint2D32f esq_dst[4])
```

Por lo tanto se nos presentan dos situaciones a la hora de transformar la imagen: la más sencilla en la que todos los datos calculados son positivos y otra en la que uno o varios vértices se encuentran en el semiplano positivo, siendo necesario rectificar la posición de las imágenes.

Todos los vértices están en el semiplano positivo

El caso más sencillo se da cuando la cámara se mueve únicamente hacia la derecha y abajo. Los vértices del siguiente frame capturado adoptarán valores positivos y no será necesario un rectificado de la posición.

En este caso la función *rectificado* sólo deberá cambiar de tamaño la imagen antigua para así adecuarla al nuevo tamaño que determina la transformación de la imagen nueva. Este proceso se realiza mediante el uso de regiones de interés, y consiste en aumentar el tamaño de la imagen un factor A en la coordenada X y B en la coordenada Y, previamente calculados (Ver Figura 4.7).

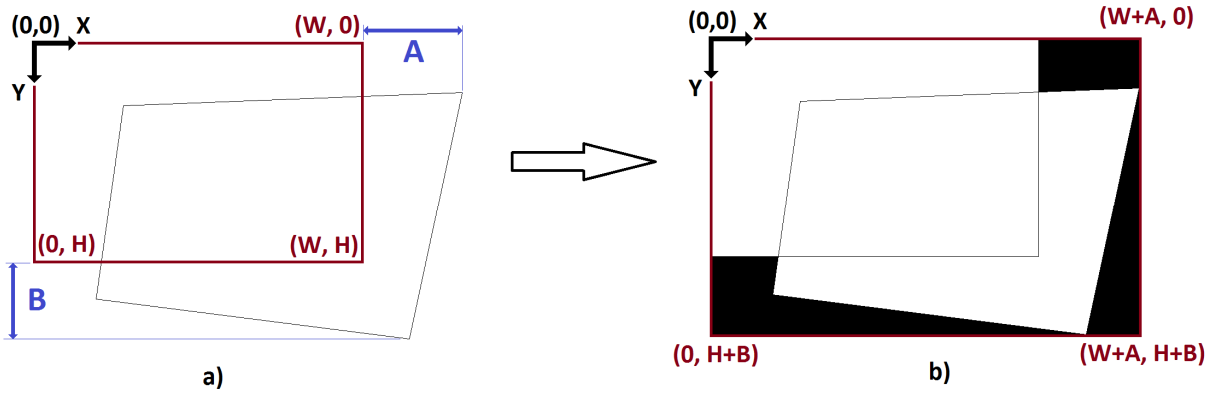


Figura 4.7: Transformación de la imagen para valores positivos.

En la Figura 4.8 se muestra el proceso que siguen tanto la imagen antigua como la nueva hasta ser fusionadas. En resumen la imagen antigua se adecúa al tamaño nuevo y la imagen nueva se transforma adecuadamente para hacerla coincidir con los vértices de destino. El tamaño de ambas imágenes es igual tras realizar el proceso, para posteriormente poder fusionarlas correctamente.

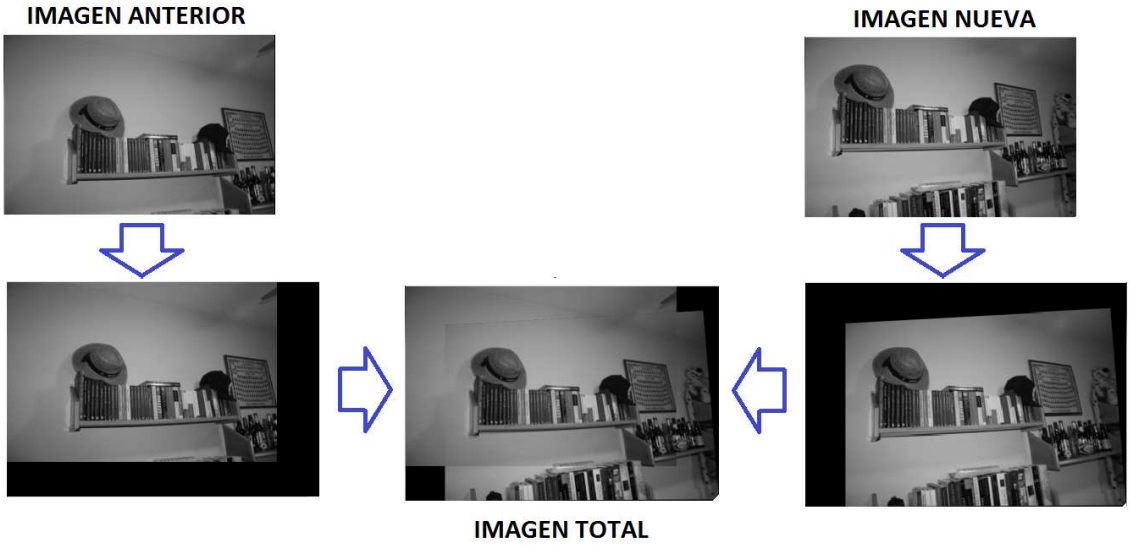


Figura 4.8: Proceso de transformación para valores positivos.

Uno o varios vértices están en el semiplano negativo

Si la cámara efectúa un movimiento hacia la izquierda y/o arriba, los valores de uno o varios de los vértices de destino adoptarán valores negativos, presentándonos el problema de que la ventana de representación sólo muestra valores ubicados en el semiplano positivo, ya que por definición su ángulo superior izquierdo se encuentra siempre en el origen de coordenadas, y variamos su tamaño únicamente con valores positivos de los ejes.

La solución propuesta a este inconveniente consiste en trasladar todas las coordenadas de cada imagen para que todos los puntos se ubiquen en el semiplano positivo. En este apartado mostramos el funcionamiento en el caso de que la imagen contenga coordenadas negativas en ambos ejes, si bien si sólo se presentan valores negativos en un solo eje el funcionamiento es el mismo, teniendo en cuenta que las correcciones se realizarían únicamente en dicho eje.

En este caso la función *rectificado* además de aumentar el tamaño de la imagen anterior, la traslada los factores necesarios A y B (que se corresponden con los valores mínimos calculados anteriormente), con el objetivo de que la nueva imagen quede en el semiplano positivo, como se muestra en la Figura 4.9.

De igual manera se modifican los vértices de destino de la transformación usando los mismos factores, de manera que los cuatro nuevos vértices de destino y, en consecuencia, las coordenadas de la imagen transformada adopten valores positivos.

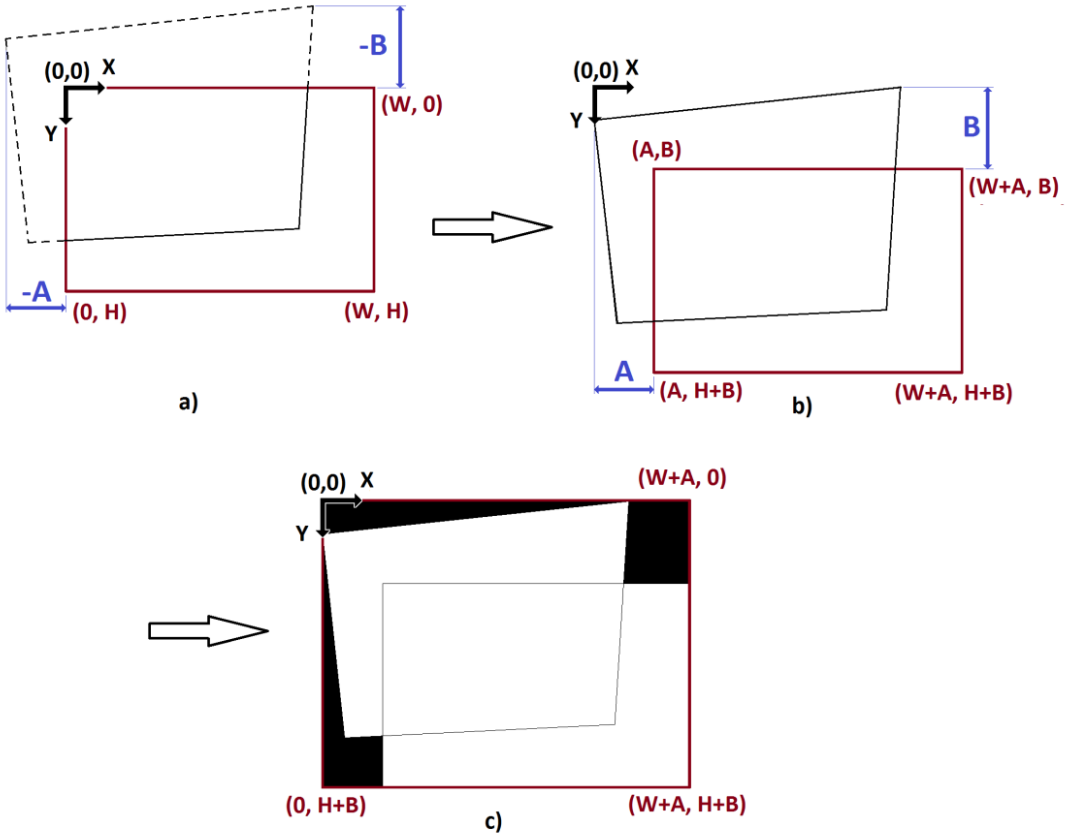


Figura 4.9: Transformación de la imagen para valores negativos, trasladando la imagen antes de fusionar.

En la Figura 4.10 se observa el proceso completo para la transformación con valores negativos. La imagen antigua se aumenta y desplaza, mientras que la nueva se transforma a los nuevos vértices trasladados al semiplano positivo.

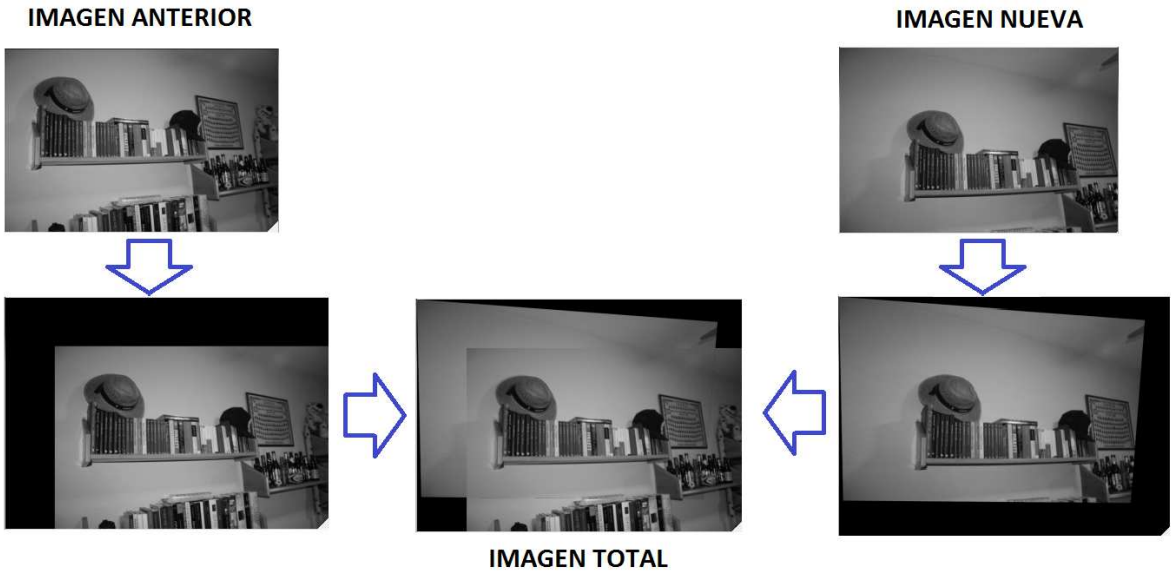


Figura 4.10: Proceso de transformación para valores negativos.

4.1.5 CUARTA FASE: OBTENCIÓN DE LA IMAGEN FINAL

Una vez tenemos ambas imágenes preparadas se procede a la fusión entre ellas. Para ello hemos implementado una función llamada *sumar*, a la que introducimos por parámetro ambas imágenes y el tamaño de éstas, y nos devuelve la imagen total.

```
IplImage* sumar( const IplImage* cor1, const IplImage* tras,
                 const CvSize tam)
```

Esta función recorre todos los píxeles de sendas imágenes en sentido horizontal, fila por fila hasta completar toda la imagen. Para cada posición se obtienen los valores de los píxeles correspondientes a cada una de las imágenes para calcular el valor del píxel de salida, que no es sino el resultado de realizar la media entre ellos. Dependiendo de los valores de los píxeles, podemos distinguir tres zonas con procedimientos distintos, como observamos en la Figura 4.11.

- ZONA A: Es la zona donde ambas imágenes se superponen entre sí. En esta zona los píxeles de ambas imágenes poseen datos de la imagen. Se realiza la media entre cada uno de ellos obteniendo un fusionado de ambas imágenes.
- ZONA B: En estas zonas hay datos de una de las imágenes pero de la otra los píxeles tienen un valor nulo (negro). La función sumará ambos valores, obteniéndose a la salida únicamente los datos de la imagen que hay presentes.
- ZONA C: En las zonas donde no hay datos de ninguna de las dos imágenes se inserta un valor nulo.

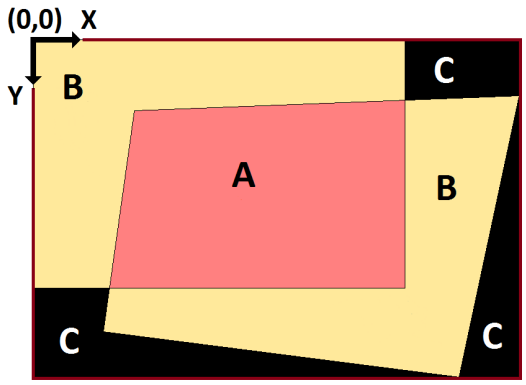


Figura 4.11: Zonas presentes en la imagen final.

Una vez realizado el fusionado, se realiza una copia de seguridad de los valores de los vértices de esta imagen total, que servirán como base para buscar nuevos puntos clave en el siguiente ciclo de funcionamiento.

BIBLIOGRAFÍA DEL CAPÍTULO 4.1

[Bra+08] [OPENCV]

4.2. MEJORAS APLICADAS

Una vez conseguidos los objetivos iniciales del programa, se han planteado una serie de mejoras para conseguir una mejora del aspecto visual y de la funcionalidad del programa.

En este apartado abarcamos las dos mejoras aplicadas al programa, que son la presentación de la imagen de salida en color y la creación de una función de detección de errores para evitar posibles fallos en el cálculo e imágenes demasiado deformadas.

4.2.1 PRESENTACIÓN DE LA IMAGEN DE SALIDA EN COLOR

La imagen capturada por la cámara debe convertirse necesariamente a escala de grises para la extracción de los puntos clave. Posteriormente realizábamos todo el proceso con la imagen monocanal obteniendo una imagen de salida en escala de grises.

Una mejora propuesta y que hemos implementado en el programa consiste en que la imagen de salida se muestre en color RGB en vez de gris, consiguiendo mejorar el aspecto visual y la información que contiene. En ocasiones las imágenes con espacio de color RGB contienen información básica que se pierde al convertirla a escala de grises, como el ejemplo que ilustra la Figura 4.12 que muestra lápices de distintos colores a los que diferencia y caracteriza su color correspondiente.

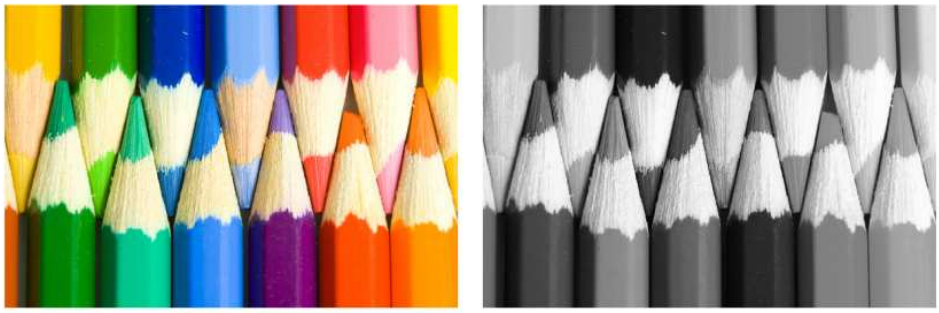


Figura 4.12: Comparación de espacios de color. En ocasiones una gran cantidad de la información que contiene una imagen reside en los colores.

Para conseguir nuestro objetivo de que la imagen de salida sea multicanal RGB debemos conservar la imagen original de tres canales y, pese a que todos los cálculos se realicen con la imagen gemela en escala de gris, al final del algoritmo usarla para representar la salida final. Por lo tanto el único tratamiento que se le aplica a la imagen en color es la transformación y fusionado final.

Tanto la función *main* como la mayoría de las funciones restantes del programa han sido modificadas para introducir esta mejora, e incluso se ha creado una nueva

llamada *rectificadoBN*, algo más sencilla que la original ahora usada para rectificar la imagen en color, y que se encarga de corregir la posición de la imagen en gris.

4.2.2 FUNCIÓN DE DETECCIÓN DE ERRORES

El hecho de que la función de detección de pares de puntos clave detecte más de 15 parejas no implica que este cálculo sea correcto. En ocasiones en las que la imagen posee un número reducido de puntos clave o éstos se encuentran en zonas que se repiten siguiendo un patrón se induce a una detección errónea de las parejas correspondientes entre las imágenes, y por lo tanto se produce un error en el cálculo de la posición de la imagen nueva.

Figura 4.13: Imagen panorámica creada por el programa que presenta diversos errores en el cálculo y la representación.

Como solución a este problema hemos programado una función que trata de detectar estos casos, y en caso de que la detección sea positiva cancele el resto de los cálculos así como el acoplamiento de la imagen. Esta función la llamamos *comprobar_errores*.

```
int comprobar_errores(const CvPoint2D32f dst[4],const int
anchoG, const int altoG, const int ancho, const int alto )
```

Debido a que las imágenes transformadas erróneamente suelen estar totalmente deformadas y tener un tamaño bastante grande, la función utiliza dos filtros para intentar detectarlas:

- El primer filtro comprueba la posición de los vértices calculada, que no debe ser superior a un cierto valor establecido respecto al del vértice antiguo, ya que la imagen que se obtendría estaría demasiado deformada respecto a la imagen central.
- El segundo filtro controla también la posición de los vértices, comprobando que cada uno de ellos tenga una posición coherente respecto a los otros, evitando así imágenes plegadas y errores de la misma naturaleza.

4.2.1.2 FILTRO DE TAMAÑO MÁXIMO

Cuando la cámara se va alejando de la primera foto central, las nuevas imágenes se van deformando progresivamente hasta un punto en el que los objetos que contiene están totalmente deformados y distorsionados respecto a sus dimensiones originales.

Este efecto no supone un error en el cálculo en sí, ya que la posición calculada es correcta y la fusión ha sido válida, pero la deformación provocada es tan grande que podría resultar perjudicial e incluso inservible para tratamientos posteriores, como se puede observar en la Figura 4.14.

Figura 4.14: Efecto de la distorsión según nos alejamos de la foto central.

Para evitar que las imágenes se deformen en exceso, hemos decidido establecer un máximo de tamaño para la nueva imagen transformada, que se ha fijado en un aumento de tamaño del 180% para el eje de abscisas, y de un 200% para el eje de ordenadas, como máximo.

4.2.1.2 FILTRO DE POSICIÓN

En ocasiones el algoritmo de detección de pares de puntos clave consigue enumerar más de 20 parejas, pero están calculadas erróneamente. Esto se debe a fallos en la interpretación de las parejas, y provoca que la posición de los nuevos vértices sea totalmente incoherente a como debería ser.

En la gran mayoría de las ocasiones en las que esto ocurre, los vértices obtenidos no siguen el orden que deberían llevar o se encuentran en posiciones totalmente atípicas, distorsionando totalmente la forma de la imagen. Estos vértices puede que compartan la misma posición o que se encuentren en posiciones imposibles, como en el ejemplo representado en la Figura 4.15.

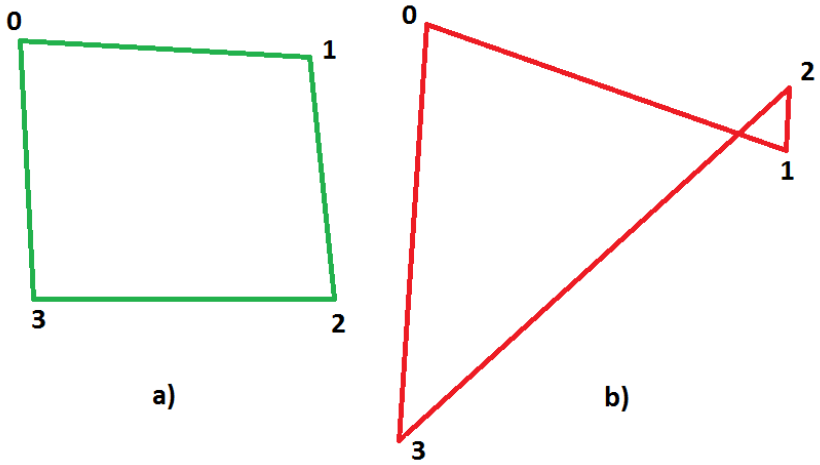


Figura 4.15: Ejemplo de la coherencia de los vértices.
a) Posición de los vértices coherente. b) Posición incoherente

La inclusión de una imagen con este fallo en la imagen compuesta puede tener consecuencias desastrosas, ya que como se observa en la Figura 4.16 la inserción de una imagen de estas características daña totalmente el resto de la imagen, imposibilitando los cálculos de los posteriores ciclos de funcionamiento del programa. Por este motivo la presencia de esta función de detección de errores se torna en fundamental para el correcto funcionamiento del programa.

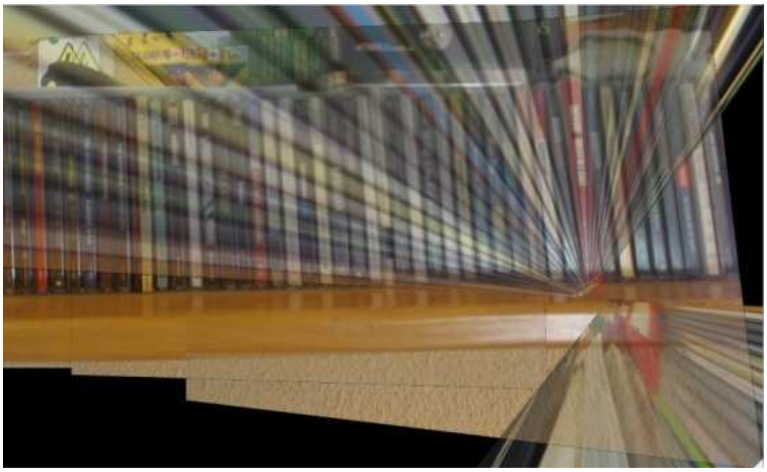


Figura 4.16: Imagen en la que se ha insertado una foto cuya posición ha sido erróneamente calculada.

BIBLIOGRAFÍA DEL CAPÍTULO 4.2

[Bra+08] [OPENCV]

CAPÍTULO 5

CONCLUSIONES

5.1. RESULTADOS EXPERIMENTALES

Los resultados experimentales obtenidos demuestran el buen funcionamiento de la aplicación. Hemos creado panorámicas de una nitidez y calidad excelente capturando escenas con los suficientes objetos para el reconocimiento de un número adecuado de puntos clave.

Se han conseguido imágenes panorámicas de un tamaño del orden de diez veces superior al tamaño de la primera foto tomada con la cámara web, por lo que se consigue ampliar notablemente el campo de visión de la escena que se desea capturar.

Figura 5.1: Ejemplo de panorámica creada con el programa.

En la Figura 5.1 se muestra un ejemplo de una panorámica creada con el programa. Se puede observar que la unión entre la mayoría de las imágenes es bastante correcta, y que cuanto más cercanos se encuentren los objetos al objetivo de la cámara, más tienden a deformarse por el efecto producido.

A continuación en las siguientes Figuras se presentan otras cinco panorámicas creadas con el programa y que serán usadas para el estudio de tiempos del Apartado 5.2.

Figura 5.2: Panorámica 1.

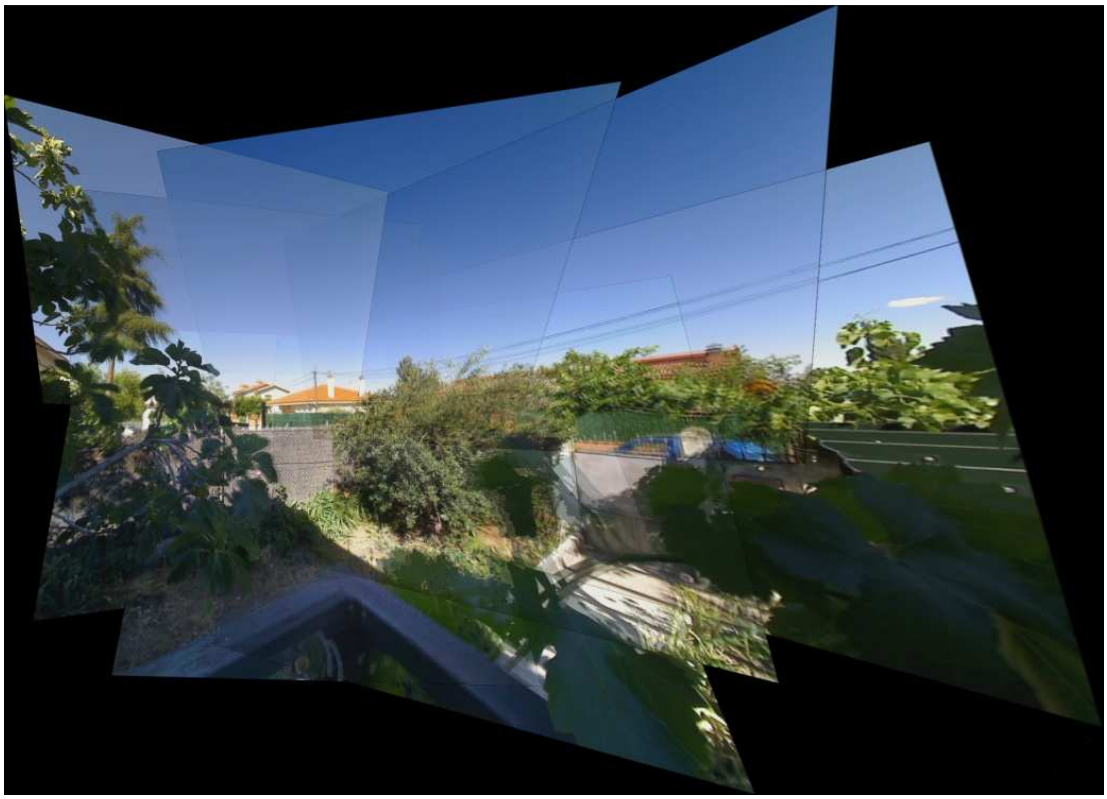


Figura 5.3: Panorámica 2.

Figura 5.4: Panorámica 3.

Figura 5.5: Panorámica 4.

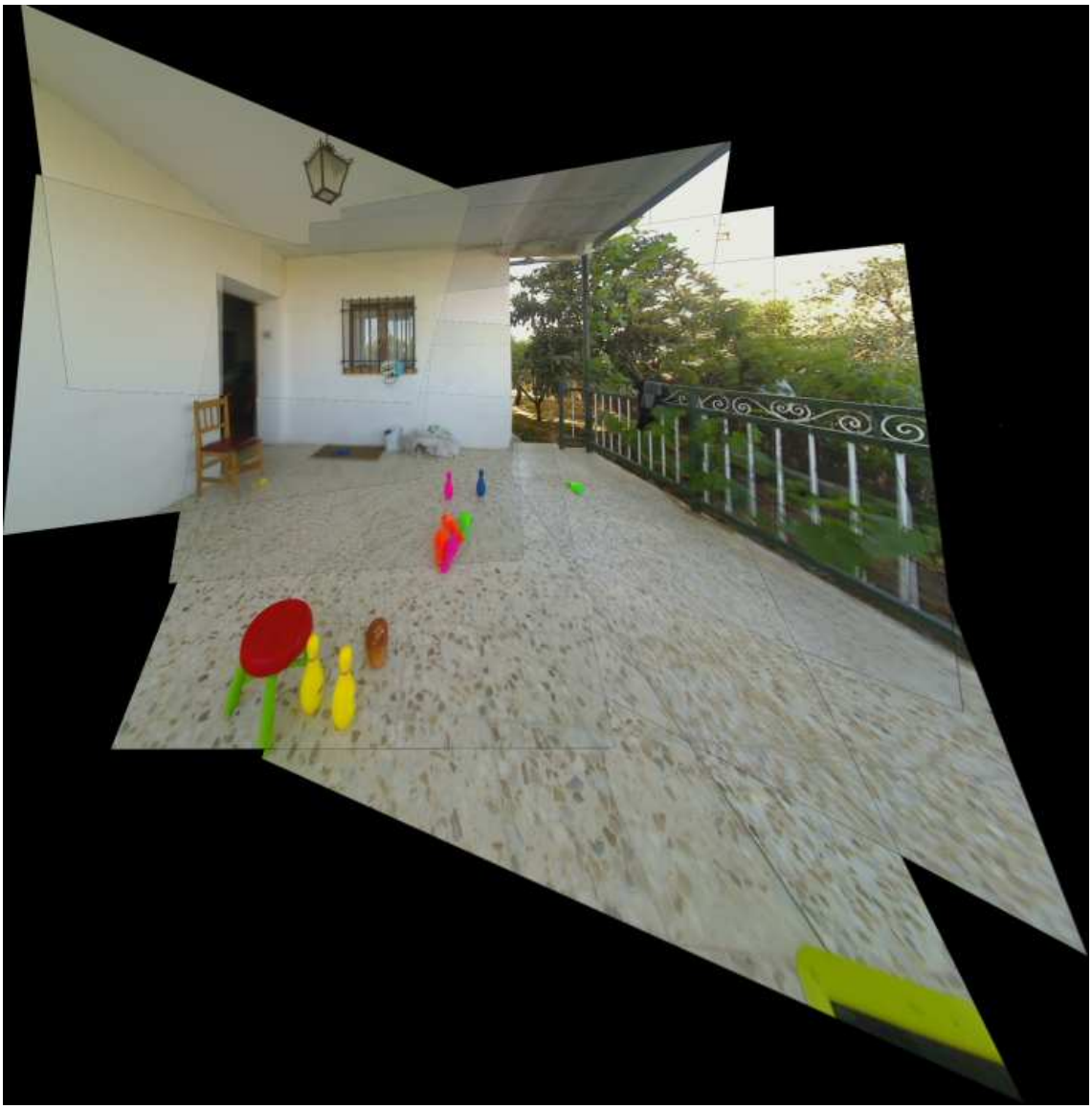


Figura 5.6: Panorámica 5.

Observando los resultados se puede concluir que el programa obtiene unas panorámicas sobresalientes y, como veremos más adelante, empleando unos tiempos más que razonables. En entornos en los que de la escena capturada el programa extrae un número reducido de puntos clave, el funcionamiento del programa se ve entorpecido.

Cabe destacar también que lógicamente el programa está diseñado para trabajar en entornos estáticos. Como es de suponer en entornos dinámicos los objetos que se desplacen aparecerán en varios frames capturados, o bien se verán borrosos, efecto que se puede apreciar claramente en la Figura 5.3, donde muchas de las hojas de los árboles se aprecian borrosas debido al movimiento provocado por el viento.

5.1.1 ESTUDIO DEL MOVIMIENTO DE LA CÁMARA

Un robot puede mover su sistema de visión de varias formas posibles. En este apartado estudiamos dos posibles movimientos de la cámara: el movimiento de rotación y el de traslación. En analogía con los humanos correspondería a mirar girando la cabeza o bien caminando manteniendo la cabeza inmóvil.

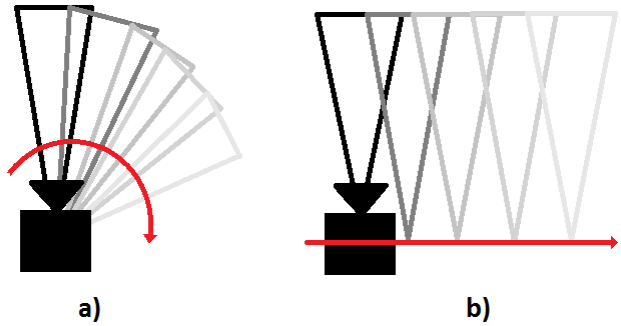


Figura 5.7: Posibles movimientos de la cámara. a) Rotación. b) Traslación.

En la Figura 5.7 se ilustran los dos movimientos de la cámara a estudiar. Como veremos a continuación, el movimiento que más nos beneficia para nuestros objetivos es el de giro, por lo que procuraremos que nuestra cámara se mueva siguiendo un giro sobre un eje.

5.1.1.1 MOVIMIENTO DE GIRO

Cuando el movimiento de la cámara es de giro, la imagen se deforma en forma de campana, quedando las zonas centrales en su forma original y cada vez deformándose más según nos alejamos de la imagen central.

Figura 5.8: Distorsión provocada por un giro horizontal de la cámara.

Este movimiento es el que resulta más adecuado para nuestro programa, ya que, como podemos observar en la Figura 5.8, el acoplado de las imágenes se realiza de manera correcta, en comparación con los resultados obtenidos en el siguiente apartado, que estudia las imágenes obtenidas con un movimiento de traslación de la cámara.

Ángulo de giro

El ángulo que gire la cámara determinará cuánto se superpondrán las imágenes. Esta superposición deberá ser lo mayor posible ya propiciará un mayor número de puntos clave en común entre las dos imágenes, lo que supondrá un mejor acoplamiento entre ellas. Por lo tanto podemos concluir que cuanto menor sea el ángulo girado por la cámara, posiblemente se producirá un mejor acoplamiento entre las consecutivas imágenes.

Además, gracias a la restricción del tamaño de la nueva imagen, el ángulo girado no debe ser un valor muy alto, ya que cuanto más haya girado la cámara más grande será la siguiente imagen. Si el ángulo girado es lo suficientemente grande es posible que la deformación provocada exceda los límites establecidos en la función de detección de errores, y ésta imposibilite que el programa realice los demás cálculos.

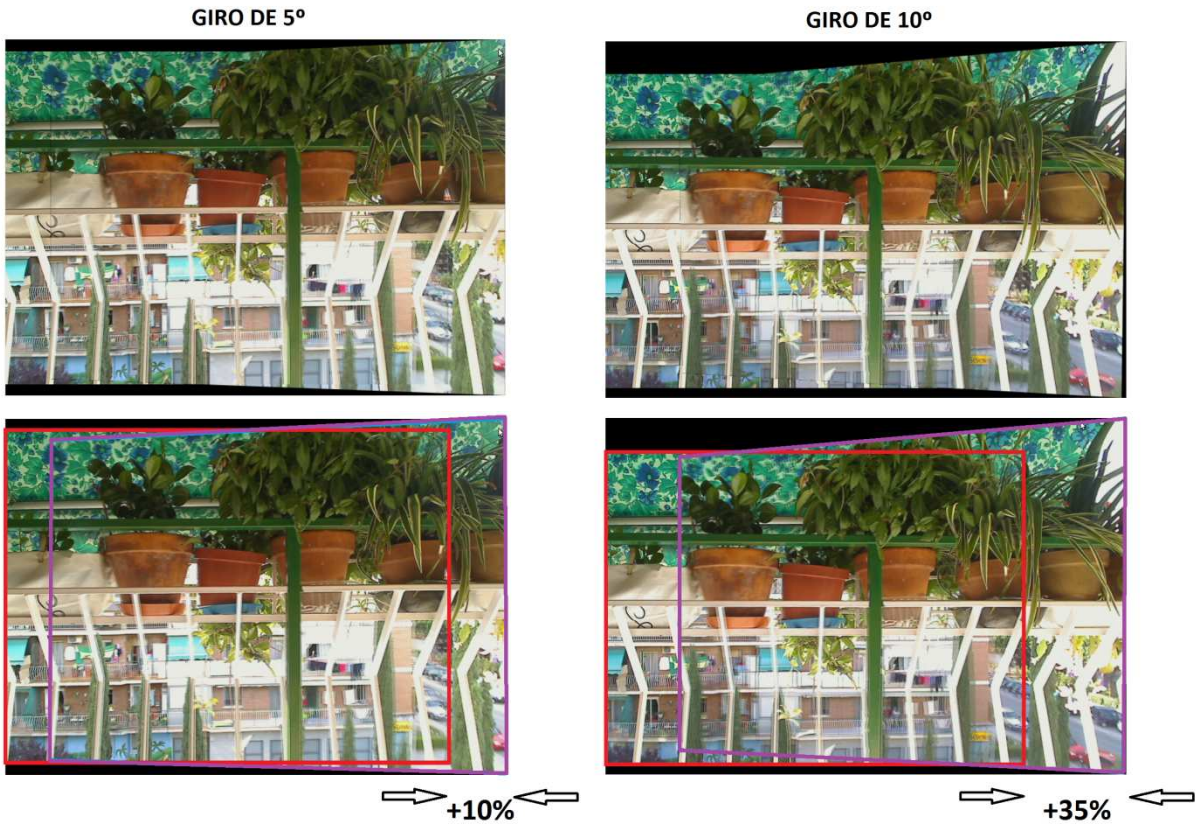


Figura 5.9: Cambio de tamaño dependiendo de la rotación. A la izquierda se observa un giro de 5º, produciéndose un aumento de tamaño del 10%. A la derecha el giro es de 10º, con un aumento de tamaño del 35%.

El ángulo de giro de la cámara entre dos capturas consecutivas depende de la velocidad de giro y de la frecuencia con la que se toman las capturas. Controlando ambas variables podremos ser capaces de obtener capturas adecuadas para formar la nueva imagen.

5.1.1.3 MOVIMIENTO DE TRASLACIÓN

El caso menos favorable para nuestros intereses se da si el robot se desplaza a la vez que captura las imágenes. En esa situación es bastante probable que la calidad de la imagen panorámica sea sustancialmente menor, ya que las imágenes se deforman de manera contraria a como lo hacen en el movimiento de giro.

En este caso, la parte más deformada a la foto es la cercana a la anterior captura, que precisamente es la que menos deformación presenta, resultando una correspondencia entre ambas imágenes bastante pobre.

Figura 5.10: Distorsión provocada por la traslación horizontal de la cámara.

En la Figura 5.10 podemos observar el resultado de un movimiento de traslación puro de la cámara, en la cual se aprecia el efecto negativo que produce sobre la imagen final. Las imágenes transformadas apenas coinciden con las anteriores produciéndose un efecto de duplicado de los objetos presentes en la imagen.

En conclusión, debemos evitar el movimiento de traslación a la hora de capturar los sucesivos frames que queremos fusionar, y procurar que el único movimiento que realice la cámara mientras se efectúa la creación de la imagen panorámica sea el movimiento de giro.

5.1.2. ESTUDIO DEL TIEMPO DE EJECUCIÓN

En el presente apartado se incluyen los resultados del estudio sobre los tiempos empleados para la ejecución de cada una de las partes del programa.

5.1.2.1 CARGAS DE TIEMPO

La primera característica sometida a estudio es la carga de tiempo de cada una de las funciones del programa respecto al tiempo de ejecución total, con el objetivo de conocer cuáles son las que más ralentizan el programa y, a ser posible, intentar reducir sus tiempos.

Para ello se ha adaptado el programa para que presente por pantalla el tiempo empleado en el funcionamiento de cada función, recopilando todos los datos de tiempo. Posteriormente se han creado 5 imágenes panorámicas, formadas por 10 imágenes cada una, que se recogen en las Figuras 5.2, 5.3, 5.4, 5.5 y 5.6.

Los resultados se presentan en la Tabla 1, que recoge los valores de las medias de las 50 muestras de tiempo obtenidas para cada función. Los datos completos del estudio se pueden consultar en el Anexo 1 de la presente memoria.

FUNCIÓN	***TIEMPO***	***PORCENTAJE***
Captura de la imagen	2,068 ms	0,1087 %
Extracción de los puntos clave SURF	1368,2 ms	71,9385 %
Localización de la imagen	135,88 ms	7,1445 %
Cálculos Auxiliares	0,0011 ms	0,000001 %
Detección de errores	0,0007 ms	0,000001 %
Encuadre de la nueva imagen	0,001 ms	0,000001 %
Rectificado de las imágenes	8,1896 ms	0,4306 %
Transformación de la nueva imagen	59,046 ms	3,1046 %
Suma de ambas imágenes	287,57 ms	15,12 %
Cálculos Finales	40,945	2,1529 %
TOTAL	1901,9 ms	100%

Tabla 1: Datos de tiempo medio de ejecución por función.

Los datos se representan gráficamente en la Figura 5.11. En la tabla no se incluye el tiempo de inicialización de las variables, que lógicamente se consume al iniciar el programa y que hemos cuantificado en 339,43 ms de media.

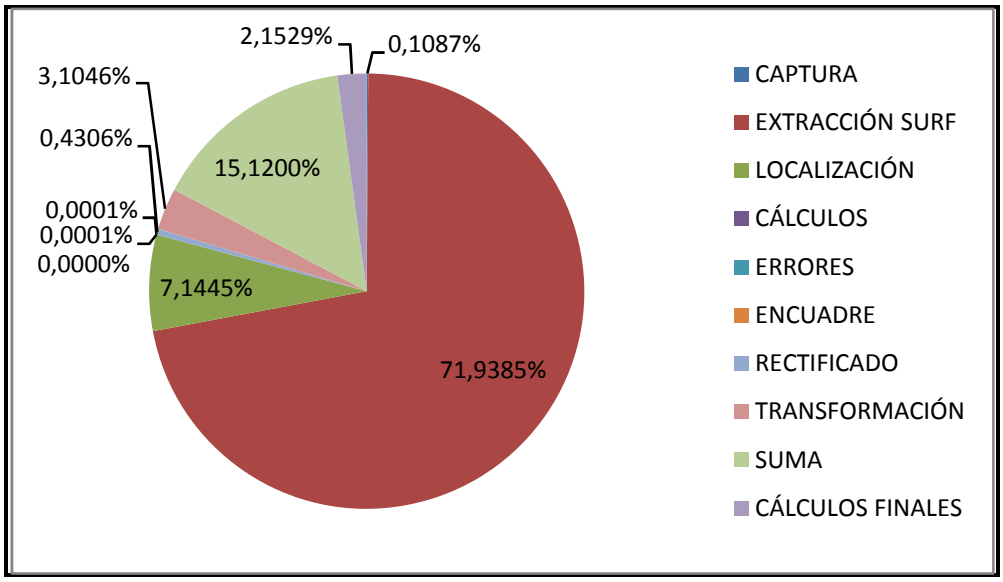


Figura 5.11: Gráfico de sectores que representa el tiempo medio de ejecución por función.

Como podemos observar, la extracción de los puntos clave SURF es claramente la función del programa que más tiempo necesita, prácticamente el 72% del tiempo total de ejecución de media. La suma de las imágenes ocupa el 15% del tiempo y la localización un 7%.

En caso contrario vemos que hay funciones que se realizan casi instantáneamente. Es el caso de la captura de la imagen por parte de la cámara, los cálculos auxiliares, el encuadre, rectificado y transformación de la imagen. Además la función introducida para la detección de errores en las imágenes calculadas trabaja en sólo unos microsegundos.

Basándonos en las etapas descritas en el apartado 4, obtenemos los resultados representados en la Tabla 2:

FUNCIÓN	***TIEMPO***	***PORCENTAJE***
1ª Fase: Captura de la imagen	2,068 ms	0,11 %
2ª Fase: Análisis de la imagen	1504,1 ms	79,08 %
3ª Fase: Transformación de la imagen	67,237 ms	3,54 %
4ª Fase: Suma de ambas imágenes	328,51 ms	17,27 %
TOTAL	1901,9 ms	100%

Tabla 2: Datos de tiempo medio de ejecución por fase de funcionamiento.

El diagrama de sectores se representa en la Figura 5.12. Obviamente la tercera fase de análisis de la imagen abarca la gran mayoría del tiempo, concretamente el 79% del total.

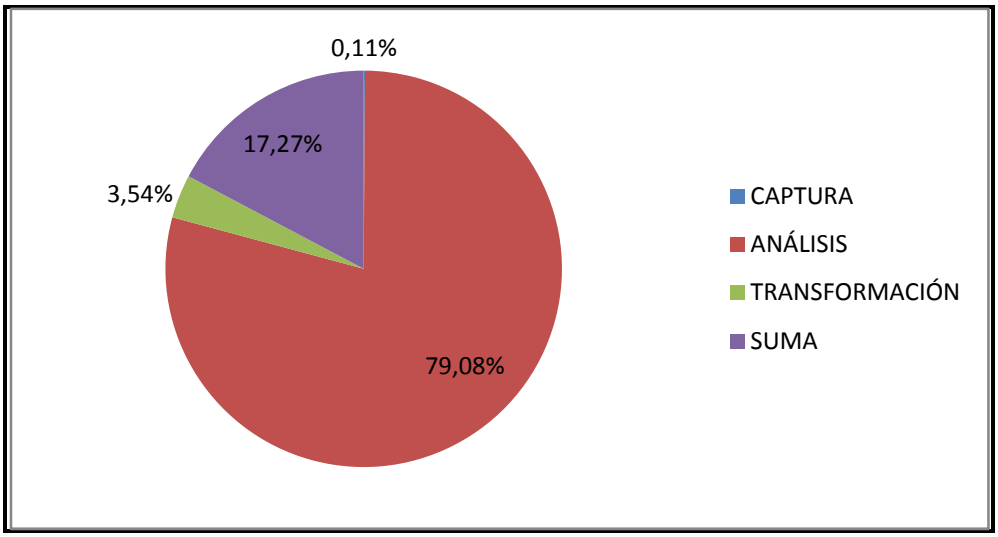


Figura 5.12: Gráfico de sectores que representa el tiempo medio de ejecución por bloque de funcionamiento.

5.1.2.2 DEPENDENCIA DEL TAMAÑO

Los datos recogidos anteriormente reflejan la media del tiempo que tarda el programa en fusionar varias imágenes pero, como es lógico, cuanto más grande vaya resultando la imagen total panorámica, el programa emplea más tiempo en realizar los cálculos. En este apartado vamos a comprobar cómo afecta el tamaño de la imagen a las distintas funciones implicadas en los cálculos a partir de las tablas recogidas en el Anexo 1. En la Figura 5.13 se representa el tiempo total de ejecución respecto al tamaño de la imagen. Se puede observar claramente como ambas variables son directamente proporcionales.

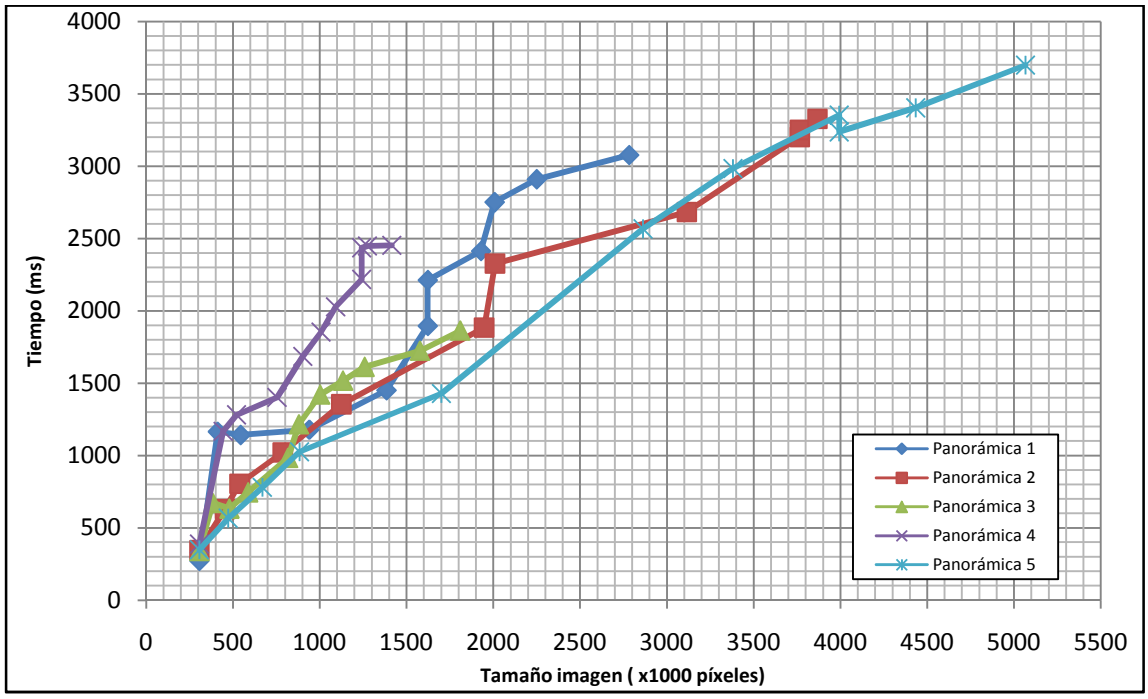


Figura 5.13: Tiempo total de ejecución respecto al tamaño de la imagen.

Vamos a analizar la respuesta de cada función. La primera función a estudio es la captura de la imagen a través de la cámara. Como era de esperar los valores obtenidos no varían sustancialmente según aumenta el tamaño de la imagen obtenida, sino que más bien varían aleatoriamente entre 1 y 4 milisegundos, como se puede observar en la Figura 5.14.

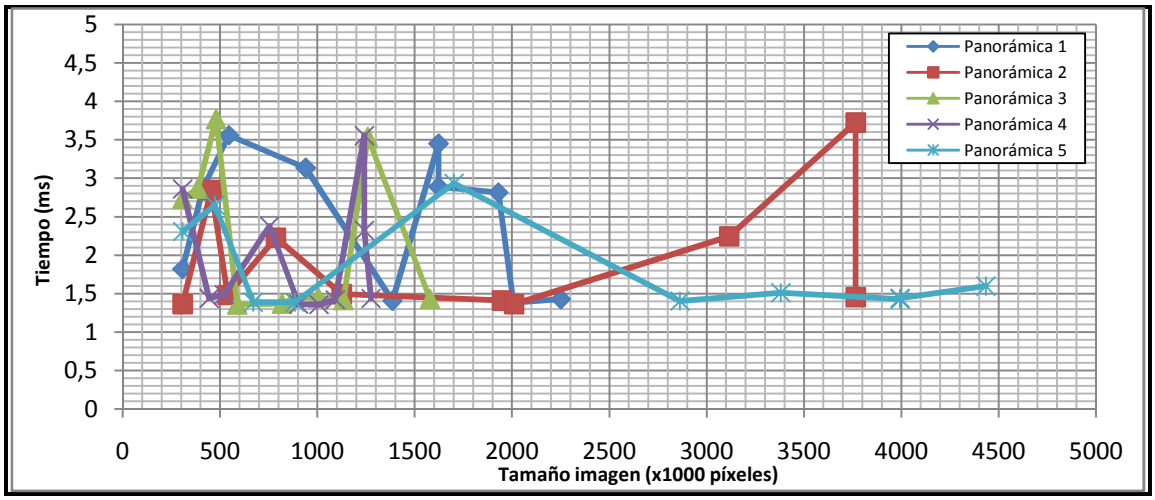


Figura 5.14: Tiempo empleado en la captura de la imagen respecto al tamaño de la imagen fusionada anterior.

El tiempo destinado a la extracción de los puntos clave SURF es dependiente del tamaño de la imagen y de los puntos clave que contenga ésta. Pese a esta doble dependencia, como se observa en la Figura 5.15 la gráfica tiempo-tamaño se comporta de una manera prácticamente lineal, ya que cuánto más grande sea la imagen, probablemente posea más puntos clave.

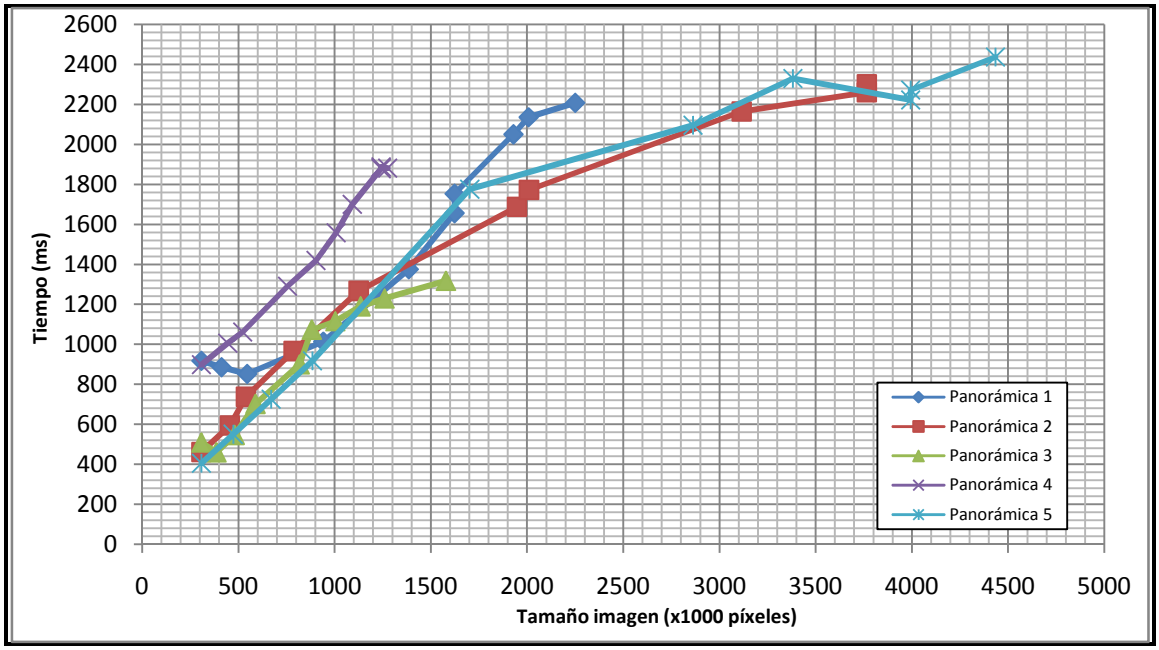


Figura 5.15: Tiempo empleado en la extracción de los puntos clave de ambas imágenes respecto al tamaño de la imagen fusionada anterior.

Un caso parecido se nos presenta en los tiempos dedicados a la localización de la imagen respecto a la anterior. Esta función también es dependiente del tamaño de la imagen aunque en menor medida que la dependencia del número de puntos clave presentes en las imágenes (Figura 5.16). El estudio de la dependencia respecto a los puntos clave presentes en las imágenes se recoge en el apartado siguiente.

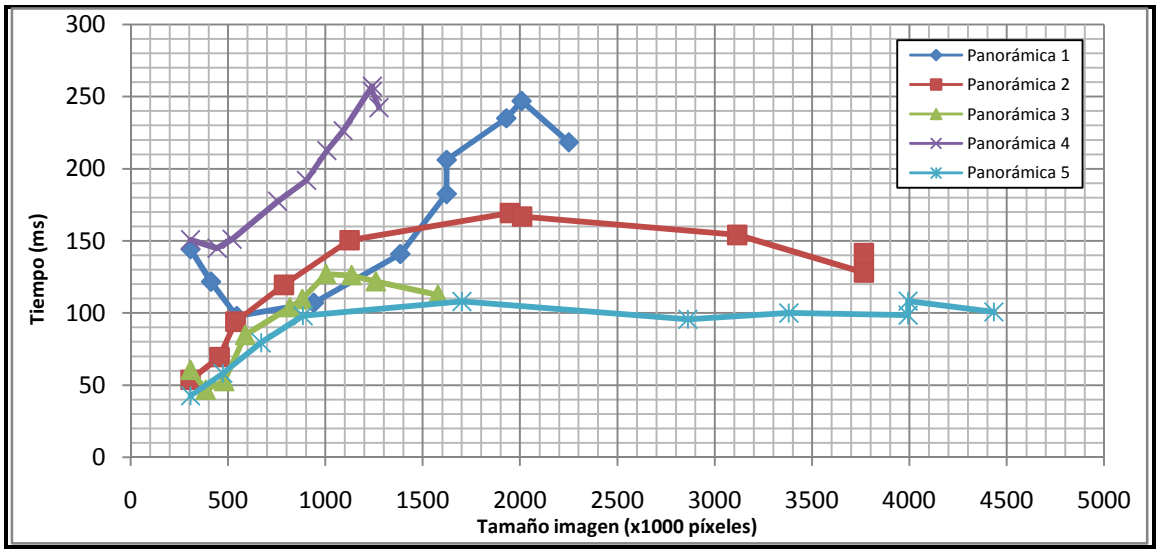


Figura 5.16: Tiempo empleado en la localización de la imagen respecto al tamaño de la imagen fusionada anterior.

Como podemos observar en la Figura 5.17, el rectificado de la imagen es la única función que depende principalmente del tamaño de ésta, por lo que la gráfica obtenida presenta un comportamiento lineal. Las variaciones que se aprecian son debidas a que el tiempo utilizado por la función varía dependiendo de la presencia de vértices de destino con valores negativos, en cuyo caso tardaría ligeramente más. Aun así, estas variaciones son mínimas.

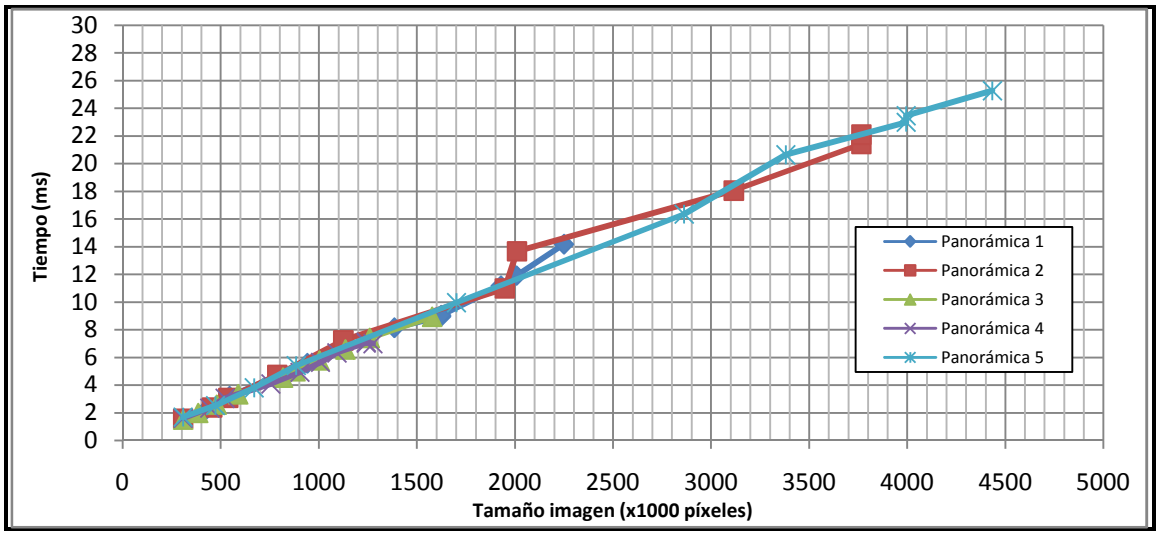


Figura 5.17: Tiempo empleado en el rectificado de las imágenes respecto al tamaño de la imagen fusionada anterior.

Las siguientes funciones estudiadas son la de transformación de la imagen nueva, la suma de ambas imágenes y los cálculos finales que incluyen la salida por pantalla y guardado de la imagen, entre otros. Las tres funciones se comportan de manera bastante similar, ya que todas dependen tanto del tamaño de la imagen vieja como de la posición de la nueva imagen, de ahí su gráfica poco lineal. Se representan en las Figuras 5.18, 5.19 y 5.20.

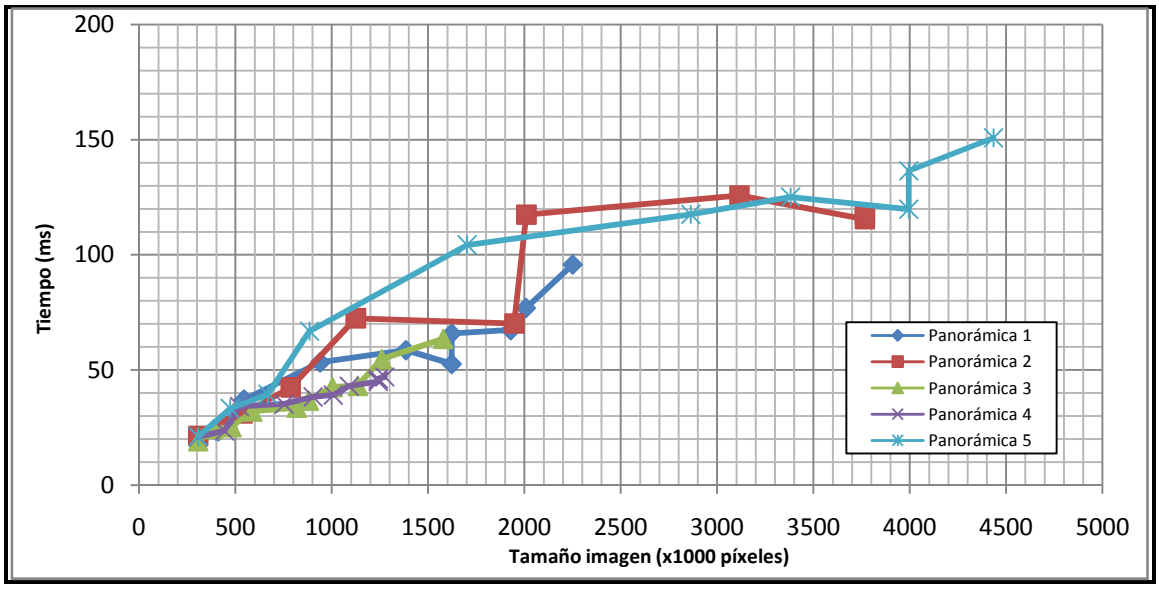


Figura 5.18: Tiempo empleado en la transformación de la imagen nueva respecto al tamaño de la imagen fusionada anterior.

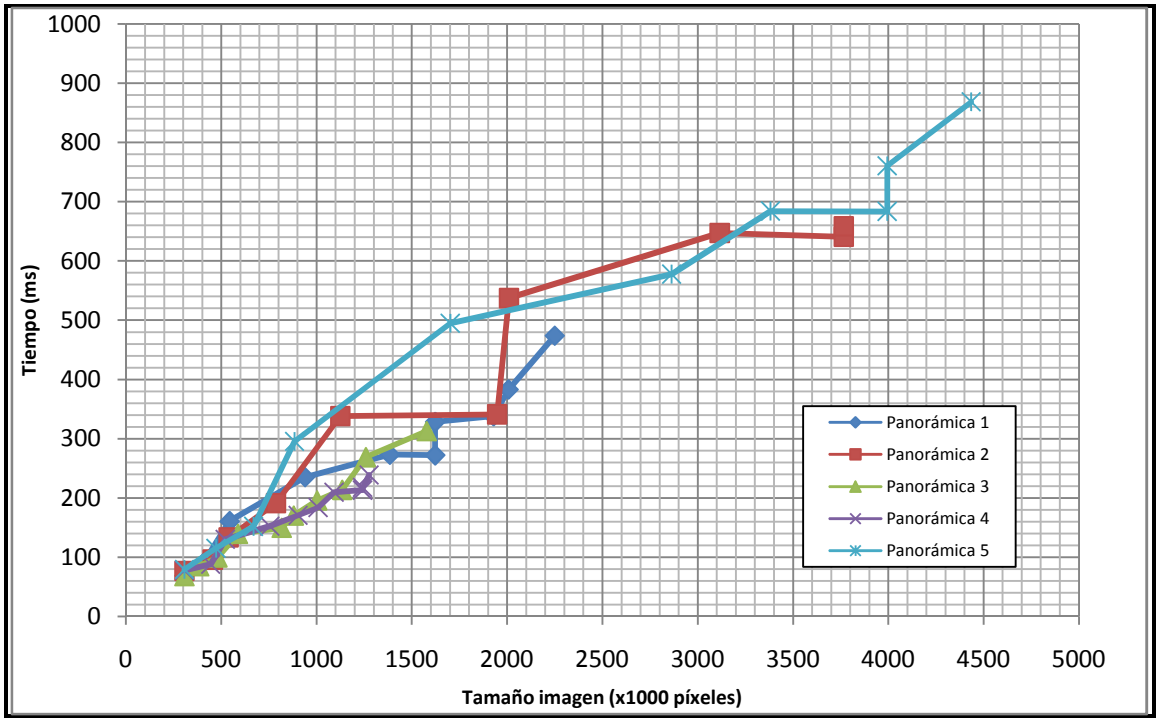


Figura 5.19: Tiempo empleado en la suma de las imágenes respecto al tamaño total.

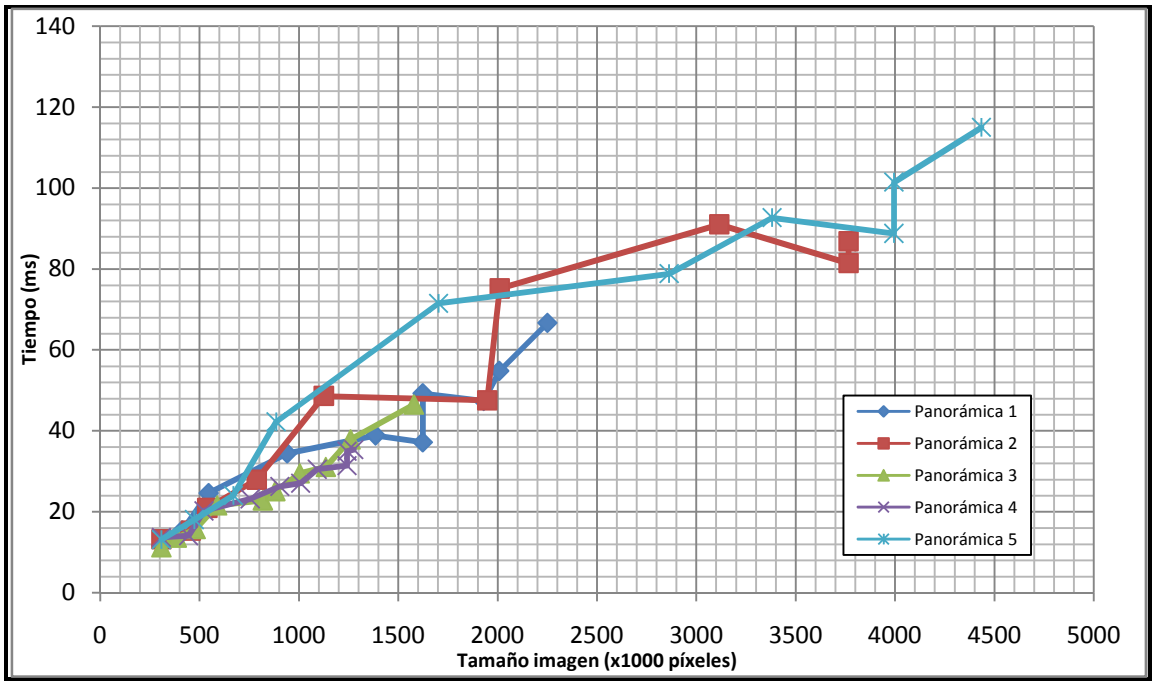


Figura 5.20: Tiempo empleado en los cálculos finales respecto al tamaño total de la imagen.

El tiempo empleado por el resto de las funciones es tan pequeño, que hace que sea prácticamente inútil su estudio, por lo que se han obviado.

5.1.2.2 DEPENDENCIA DEL NÚMERO DE PUNTOS CLAVE

Una vez que hemos visto cómo el tiempo de funcionamiento de las distintas funciones varía dependiendo del tamaño de la imagen vieja, vamos a estudiar cómo se comportan las funciones dependientes del número de puntos clave presentes en las imágenes. Para ello, hemos ejecutado el programa 10 veces encuadrando únicamente la primera y segunda fotografía anotando el número de descriptores de cada fotografía y el tiempo utilizado para encontrarlos. Los datos obtenidos se recogen en la Tabla 3:

NÚMERO TOTAL DE PUNTOS CLAVE	EXTRACCIÓN SURF	LOCALIZACIÓN IMÁGENES
481	289,234 ms	16,2984 ms
969	369,846 ms	30,7583 ms
1597	482,474 ms	57,6621 ms
1751	448,03 ms	54,0329 ms
2071	518,976 ms	62,2315 ms
2149	544,087 ms	68,1761 ms
2865	643,79 ms	82,2718 ms
3564	749,84 ms	106,778 ms
4036	815,15 ms	120,977 ms
4109	813,664 ms	135,887 ms

Tabla 3: Datos de tiempo de ejecución dependiendo del número total de puntos clave en ambas imágenes.

Los datos experimentales recogidos se muestran gráficamente en las Figuras 5.21 y 5.22. En ambas gráficas se distingue claramente cómo el tiempo dedicado a la ejecución de ambas funciones es directamente proporcional al número de puntos clave totales presentes en ambas imágenes.

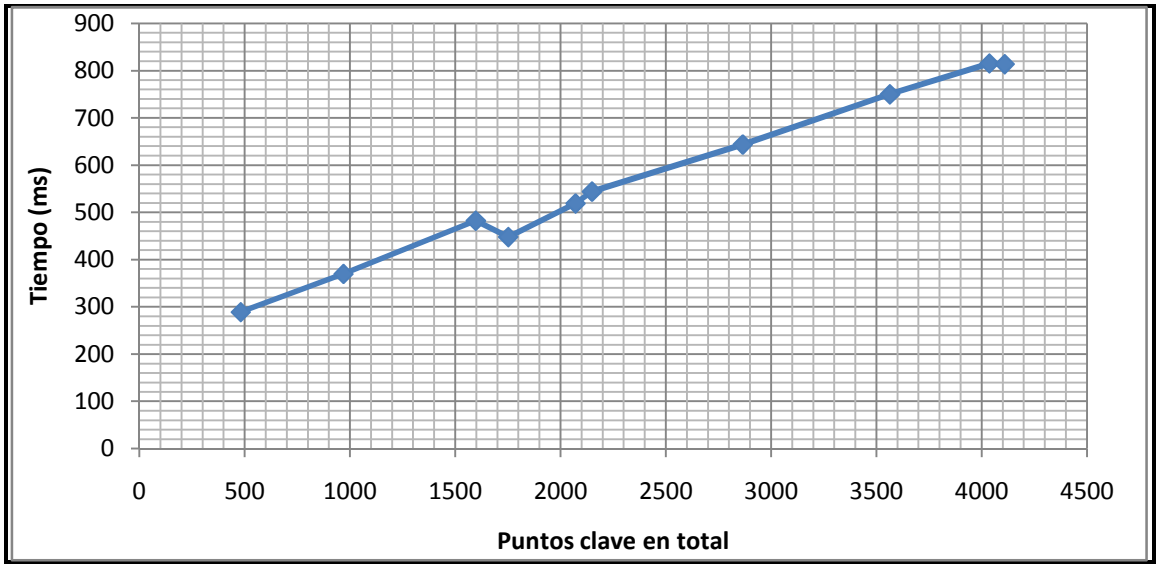


Figura 5.21: Tiempo empleado en la extracción de las características SURF respecto al número total de puntos clave presentes en ambas imágenes.

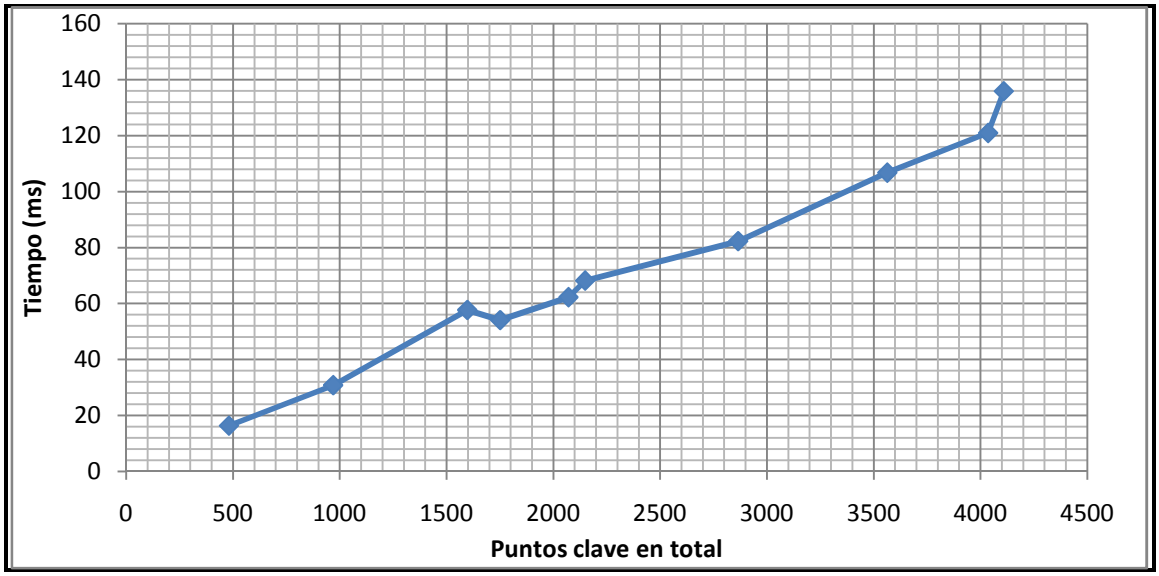


Figura 5.22: Tiempo empleado en la localización de la imagen nueva respecto al número total de puntos clave presentes en ambas imágenes.

5.2. TRABAJOS FUTUROS

En este capítulo se abordan posibles ampliaciones a realizar sobre el programa presentado. Estas mejoras se centran en la mejora de la calidad de la imagen final obtenida, con la pretensión de que ésta contenga la mayor cantidad de información posible, así como de la eliminación de las imperfecciones presentes desde su creación. Se plantean dos mejoras, la corrección de la interpolación introducida al transformar las imágenes y la diferencia de iluminación de cada una de ellas.

5.2.1 CORRECCIÓN DE LA INTERPOLACIÓN

La forma de la nueva imagen transformada es en la gran mayoría de los casos un cuadrilátero irregular cuyos lados poseen una cierta inclinación respecto a los ejes de coordenadas y que está contenida por un rectángulo. Esta inclinación de los bordes de la imagen se representa mediante métodos de interpolación.

La consecuencia de la interpolación es que al fusionar las imágenes el borde de la imagen transformada aparece en un tono oscuro, fácilmente detectable en zonas claras de la imagen como se observa en la Figura 5.23. Estos píxeles oscuros son los calculados automáticamente a través de la interpolación por la función de transformación de la imagen.

Figura 5.23: Ejemplo de fallo en la representación de los bordes de las fotos debido a la interpolación.

5.2.2 PREPROCESADO DE LA IMAGEN

En ocasiones la exposición e iluminación de una imagen capturada difiere respecto a las anteriores. Al fusionar las imágenes se aprecia esta diferencia entre ellas proporcionando a la panorámica un aspecto irreal que no beneficia a los posibles tratamientos posteriores, como puede verse en la Figura 5.24.

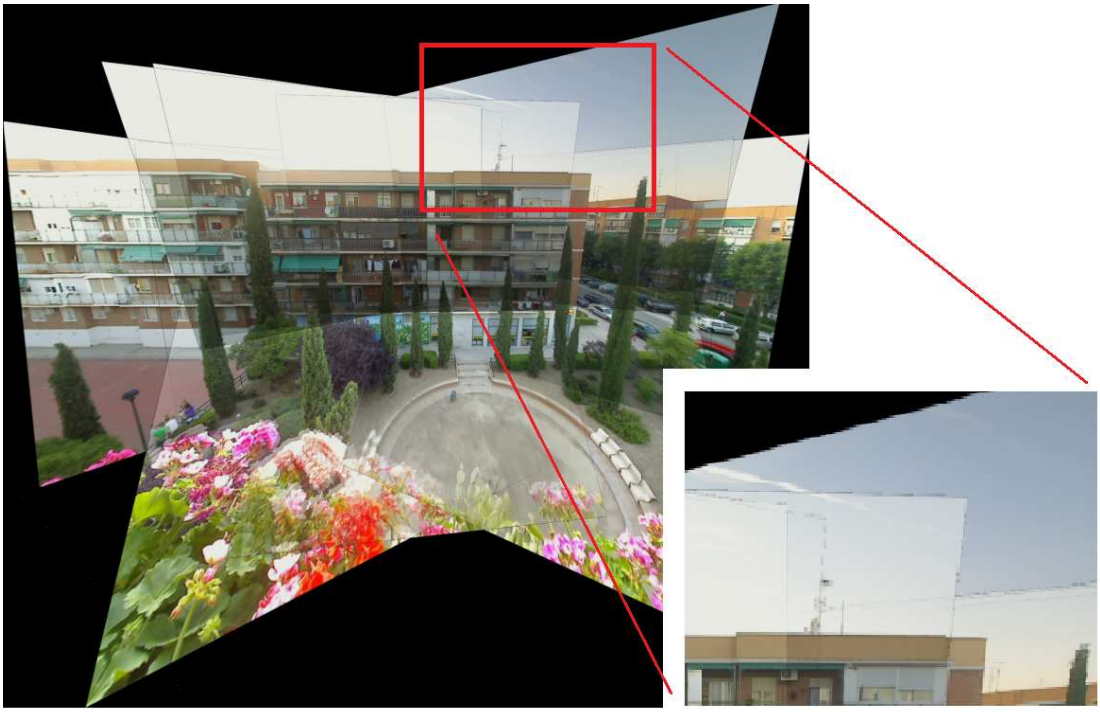
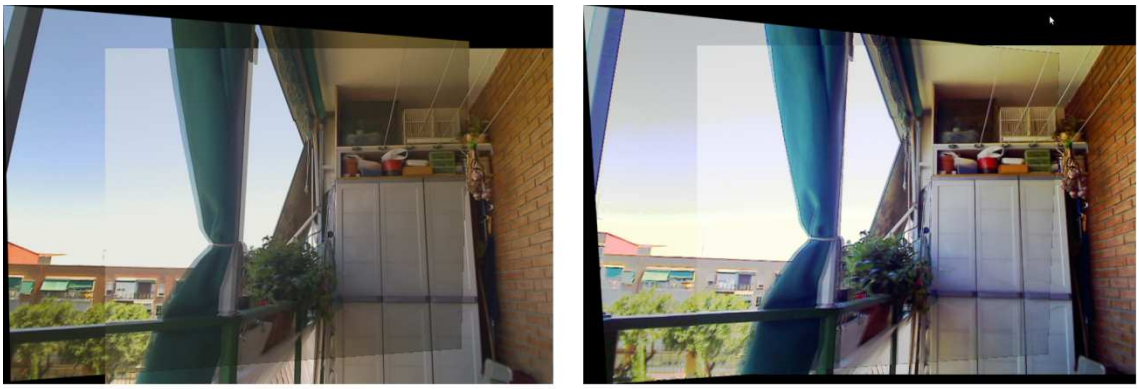


Figura 5.24: Detalle de fotos con distinta exposición en una panorámica.

Este efecto se reduciría si todas las fotos tuvieran niveles de exposición parecidos entre ellas, por lo que una posible solución sería la realización de un pre-procesado a las imágenes.

En un intento de corregir este problema se ha aplicado un ecualizado del histograma a las imágenes antes de trabajar con ellas, con el propósito de conseguir en todas ellas unos valores de exposición e iluminación parecidos, que ayudara a disminuir este efecto negativo.

Los resultados obtenidos desvelan que desgraciadamente el ecualizado soluciona el problema solamente en algunos fragmentos de la imagen, mientras que en otros la diferencia sigue siendo evidente. Además introduce ligeras variaciones en los colores, dando a la imagen un aspecto más irreal, como se muestra en la Figura 5.25, por lo que finalmente se desecho su uso.



a) b)

Figura 5.25: Comparación de resultados al aplicar un ecualizado del histograma previo. a) Imagen sin ecualizado previo. b) Imagen a la que se le ha aplicado un ecualizado previo a los cálculos.

5.3. CONCLUSIÓN FINAL

En términos generales se puede concluir que el programa cumple adecuadamente con los objetivos establecidos al inicio del proyecto. Es capaz de obtener imágenes panorámicas de escenas a través de las imágenes capturadas por una cámara web en tiempo real.

La velocidad de procesamiento es bastante buena, aunque lógicamente se va reduciendo según aumenta el tamaño de la imagen panorámica total. En cualquier caso la función de detección de errores evita que la imagen adopte un tamaño excesivamente grande, además de mejorar considerablemente el porcentaje de imágenes correctas conseguidas.

GLOSARIO

ANIMAL	AN IMAging Library
ANSI	American National Standars Institute
ATI	Array Technology Inc.
DDR	Double Data Rate
DoG	Difference Of Gaussian
FLANN	Fast Library for Approximate Nearest Neighbors search
GDDR	Graphics Double Data Rate
GIF	Graphics Interchange Format
GIMP	GNU Image Manipulation Program
GNU	GNU is Not Unix
HCI	Human Computer Interaction
IPL	Image Processing Library
IVT	Integrating Vision Toolkit
JPEG	Joint Photographic Experts Group
KIT	Karlsruhe Institute of Technology
MFSM	Modular Flow Scheduling Middleware framework
OPENCV	OPEN source Computer Vision
PCX	PiCture eXchange
PNG	Portable Network Graphics
RGB	Red Green Blue
RAM	Random Access Memory
ROI	Region Of Interest
SDRAM	Synchronous Dynamic Random Access Memory
SFM	Structure From Motion
SIFT	Scilab Image Processing
SURF	Speeded Up Robust Features
TIFF	Tagged Image File Format
USB	Universal Serial Bus
VXL	Vision *something* Library

REFERENCIAS

- [Bra+08] "*Learning openCV: computer visión with the OpenCV library*" – Gary Bradski y Adrian Kaehler. 2008 – ISBN: 9780596516130.

- [Hor86] "*Robot vision*" – Berthold Klaus Paul Horn. 1986 – ISBN 0262081598.

- [Ker+78] *"The C Programming Language"* – Brian Kernighan y Dennis M. Ritchie. 1978 – ISBN 0131101633.

- [Bay+06] *"SURF: Speeded up robust features"* - H. Bay, T. Tuytelaars y L. V. Gool. 2006

- [Low99] *"Object recognition from local scale-invariant features"* – Lowe, D. G. 1999

- [Esc01] *"Vision por Computador. Fundamentos y métodos"* – Arturo de la Escalera Hueso. 2001 – ISBN: 9788420530987

RECURSOS ELECTRÓNICOS

- [OPENCV] http://opencv.willowgarage.com/wiki/
- [SURF] http://www.vision.ee.ethz.ch/~surf/
- [AFORGE] http://code.google.com/p/aforge/
- [IVT] http://ivt.sourceforge.net/
- [IPT] http://www.mathworks.es/products/image/
- [JVis] http://javavis.sourceforge.net/
- [MFSM] http://mfsm.sourceforge.net/
- [UBUNTU] http://www.ubuntu.com/
- [GIMP] http://www.gimp.org.es/
- [SONY] http://www.sony.es/product/vn-e-series/vpcea3s1e-w/
- [LOGI] http://www.logitech.com/es-es/support/webcams/3056/
-

ANEXO 1: INFORME DETALLADO DEL ESTUDIO DE TIEMPO DE EJECUCIÓN

PANORÁMICA 1

ESTUDIO POR FUNCIONES DEL PROGRAMA

TIEMPOS DE EJECUCIÓN POR FUNCIÓN (ms)											TAMAÑO (PÍXELES)
INIC.	CAPT.	SURF	LOCAL.	CÁLC.	ERROR	ENC.	RECT.	TRANS.	SUMA	C.FINAL	
271,97	0	0	0	0	0	0	0	0	0	0	0
0	1,8192	917,32	144,32	0,0015	0,0007	0,0012	1,6081	19,334	68,916	11,798	307,2
0	2,8712	885,21	121,76	0,001	0,0007	0,0011	2,0328	23,364	91,999	15,424	411,95
0	3,5598	851,72	97,954	0,0012	0,0006	0,001	3,17	37,126	160,8	24,61	544,68
0	3,1335	1012,8	107,11	0,001	0,0007	0,001	5,5534	53,292	235,05	34,422	940,371
0	1,4057	1375,6	140,97	0,0012	0,0006	0,0009	8,1238	58,6	273,22	38,885	1384,845
0	3,4517	1656,3	182,6	0,001	0,0006	0,0009	8,9444	52,603	272,18	37,186	1622,775
0	2,8906	1752,6	206,21	0,0011	0,0006	0,001	9,0257	65,838	328,37	49,227	1622,775
0	2,8156	2050,2	235,05	0,0012	0,0007	0,001	11,153	67,326	337,96	47,387	1929,065
0	1,3816	2135,4	247	0,0013	0,0006	0,001	11,907	76,876	383,1	54,802	2007,745
0	1,4266	2207,5	218,38	0,0012	0,0007	0,0011	14,167	95,71	473,65	66,661	2249,984
											2782,56

MEDIA	2,4755	1484,5	170,14	0,0012	0,0007	0,001	7,5685	55,007	262,52	38,04
%	0,0012	0,7348	0,0842	6E-07	3E-07	5E-07	0,0037	0,0272	0,1299	0,0188

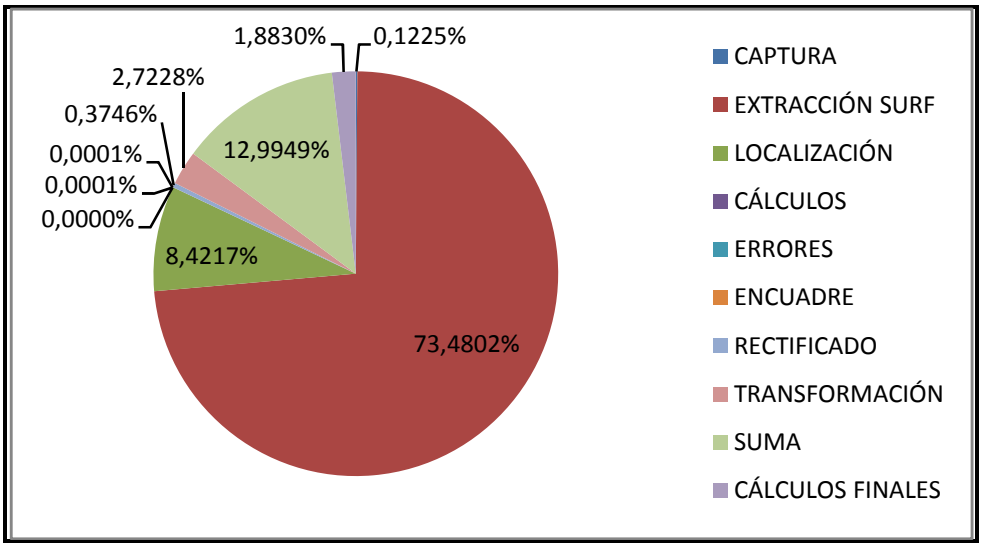


Figura A1: Porcentaje de tiempo medio usado por cada función respecto al tiempo de ejecución total, para la primera panorámica estudiada.

ESTUDIO POR BLOQUES DE EJECUCIÓN

TIEMPOS DE EJECUCIÓN POR BLOQUES (ms)					TAMAÑO (PÍXELES)
CAPTURA	ANÁLISIS	TRANSF.	SUMA	TOTAL	
1,8192	1061,6	20,943	80,714	1165,1	307,2
2,8712	1007	25,397	107,42	1142,7	411,95
3,5598	949,68	40,297	185,41	1178,9	544,68
3,1335	1119,9	58,846	269,47	1451,3	940,371
1,4057	1516,6	66,725	312,1	1896,8	1384,845
3,4517	1838,9	61,548	309,37	2213,2	1622,775
2,8906	1958,8	74,864	377,6	2414,1	1622,775
2,8156	2285,2	78,48	385,35	2751,9	1929,065
1,3816	2382,4	88,784	437,9	2910,4	2007,745
1,4266	2425,9	109,88	540,31	3077,5	2249,984
					2782,56
MEDIA	2,4755	1654,6	62,576	300,56	2020,2
%	0,0012	0,819	0,031	0,1488	

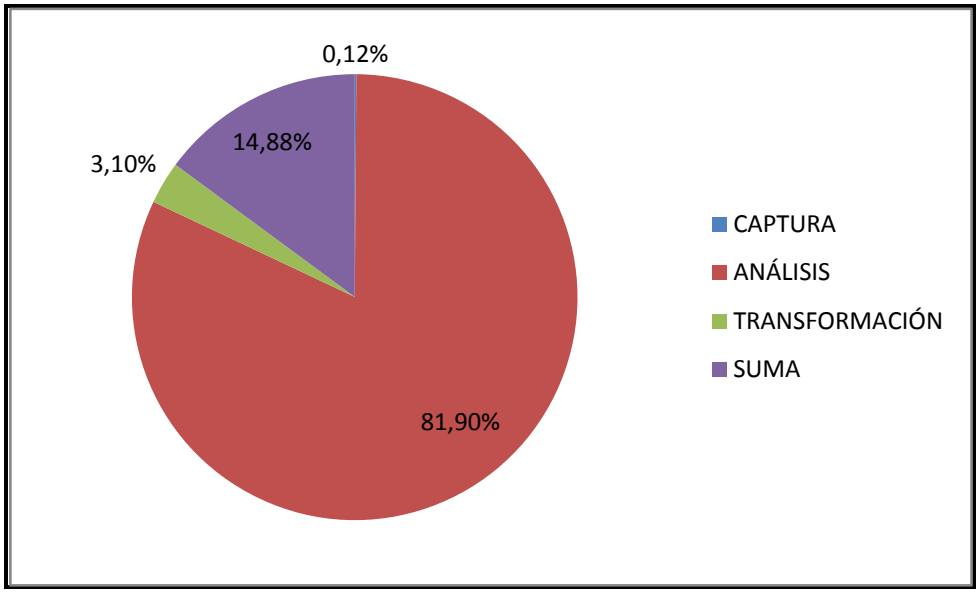


Figura A2: Porcentaje de tiempo medio usado por cada bloque de ejecución del programa respecto al tiempo de ejecución total, para la primera panorámica estudiada.

PANORÁMICA 2

ESTUDIO POR FUNCIONES DEL PROGRAMA

TIEMPOS DE EJECUCIÓN POR FUNCIÓN (ms)											TAMAÑO (PÍXELES)
INIC.	CAPT.	SURF	LOCAL.	CÁLC.	ERROR	ENC.	RECT.	TRANS.	SUMA	C.FINAL	
346,63	0	0	0	0	0	0	0	0	0	0	0
0	1,3637	461,13	53,811	0,0011	0,0009	0,001	1,5653	21,455	76,878	13,261	307,2
0	2,8467	593,22	69,596	0,0011	0,0007	0,0011	2,3698	25,468	95,938	15,356	455,511
0	1,4832	736,79	94,022	0,0009	0,0006	0,0009	3,076	31,23	133,55	20,968	538,098
0	2,228	966,1	119,59	0,0011	0,0006	0,0011	4,7401	42,474	191,29	27,959	787,15
0	1,4937	1266,5	150,57	0,0011	0,0006	0,001	7,2207	72,451	338,11	48,639	1124,504
0	1,4098	1686,5	169,36	0,0011	0,0006	0,001	10,99	70,205	341,15	47,57	1947,425
0	1,3633	1771,8	166,9	0,0009	0,0006	0,0008	13,668	117,39	537,21	75,17	2009,998
0	2,2444	2164,3	154,21	0,0009	0,0006	0,001	18,05	125,72	647,02	91,013	3115,161
0	3,7247	2260,1	128,17	0,001	0,0007	0,0009	21,424	115,44	640,65	81,464	3765,753
0	1,4547	2299,7	141,57	0,0012	0,0007	0,001	22,103	117,29	658,39	86,836	3765,753
											3867,44
MEDIA	1,9612	1420,6	124,78	0,001	0,0007	0,001	10,521	73,913	366,02	50,824	
%	0,001	0,6934	0,0609	5E-07	3E-07	5E-07	0,0051	0,0361	0,1787	0,0248	

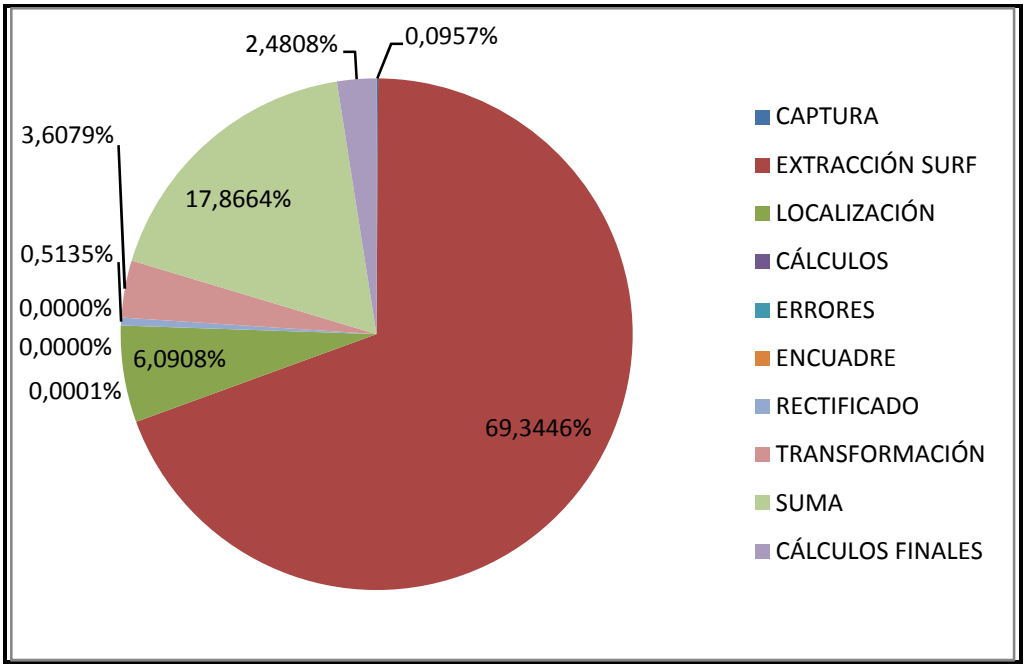


Figura A3: Porcentaje de tiempo medio usado por cada función respecto al tiempo de ejecución total, para la segunda panorámica estudiada.

ESTUDIO POR BLOQUES DE EJECUCIÓN

TIEMPOS DE EJECUCIÓN POR BLOQUES (ms)					TAMAÑO (PÍXELES)
CAPTURA	ANÁLISIS	TRANSF.	SUMA	TOTAL	
1,3637	514,95	23,021	90,139	629,47	307,2
2,8467	662,82	27,838	111,29	804,8	455,511
1,4832	830,81	34,307	154,52	1021,1	538,098
2,228	1085,7	47,215	219,25	1354,4	787,15
1,4937	1417,1	79,673	386,74	1885	1124,504
1,4098	1855,9	81,195	388,72	2327,2	1947,425
1,3633	1938,7	131,06	612,38	2683,5	2009,998
2,2444	2318,5	143,77	738,03	3202,5	3115,161
3,7247	2388,3	136,87	722,12	3251	3765,753
1,4547	2441,3	139,4	745,23	3327,3	3765,753
					3867,44
MEDIA	1,9612	1545,4	84,435	416,84	2048,6
%	0,001	0,7544	0,0412	0,2035	

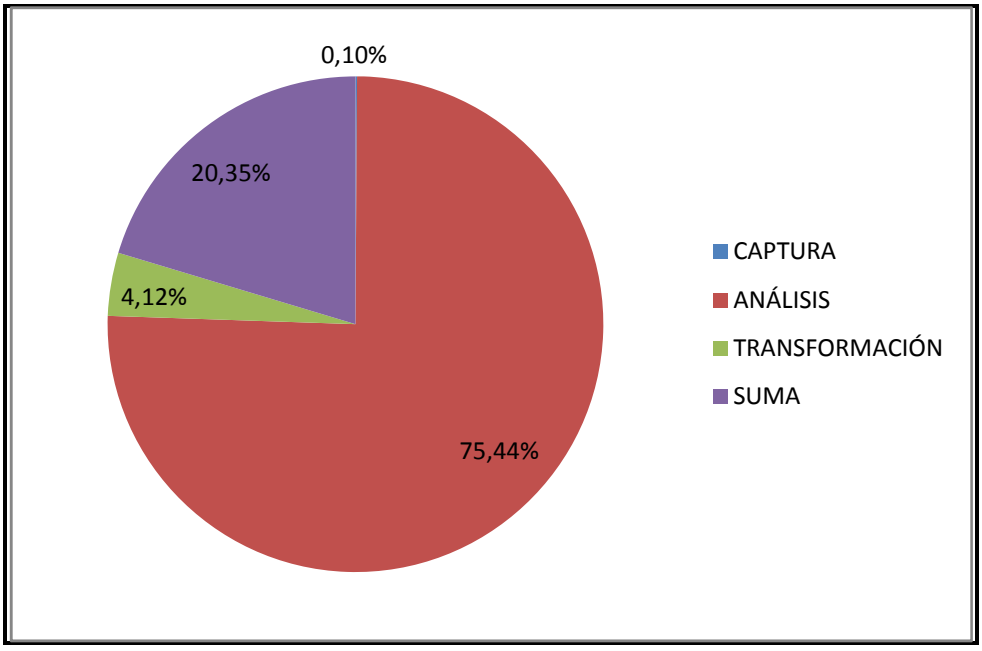


Figura A4: Porcentaje de tiempo medio usado por cada bloque de ejecución del programa respecto al tiempo de ejecución total, para la segunda panorámica estudiada.

PANORÁMICA 3

ESTUDIO POR FUNCIONES DEL PROGRAMA

TIEMPOS DE EJECUCIÓN POR FUNCIÓN (ms)											TAMAÑO (PÍXELES)
INIC.	CAPT.	SURF	LOCAL.	CÁLC.	ERROR	ENC.	RECT.	TRANS.	SUMA	C.FINAL	
341,34	0	0	0	0	0	0	0	0	0	0	0
0	2,7309	508,65	60,487	0,001	0,0006	0,001	1,5005	19,142	68,127	11,267	307,2
0	2,8631	459,67	46,84	0,001	0,0014	0,0013	1,9834	24,375	84,623	13,654	385,25
0	3,7657	546,86	52,724	0,0014	0,0007	0,0012	2,583	25,237	99,636	15,769	479,814
0	1,3586	701,85	84,745	0,0009	0,0006	0,0008	3,3111	32,071	139,31	21,626	588
0	1,3716	899,82	103,98	0,0011	0,0008	0,0009	4,5149	33,687	149,5	22,852	817,71
0	1,4469	1073,3	109,6	0,001	0,0006	0,0007	4,9827	36,763	170,16	25,136	880,76
0	1,4473	1116	127,03	0,0012	0,0009	0,0011	5,7783	42,638	195,7	29,535	1003,472
0	1,415	1190,4	126,19	0,0011	0,0006	0,0008	6,5481	43,2	213,63	31,115	1134,688
0	3,5336	1229,7	121,93	0,0011	0,0006	0,001	7,3872	54,767	268,61	37,906	1259,296
0	1,4328	1317,6	112,53	0,0011	0,0006	0,001	8,9083	63,554	312,56	46,553	1578
											1809,966
MEDIA	2,1366	904,38	94,605	0,0011	0,0007	0,001	4,7497	37,543	170,19	25,541	
%	0,0017	0,7298	0,0763	9E-07	6E-07	8E-07	0,0038	0,0303	0,1373	0,0206	

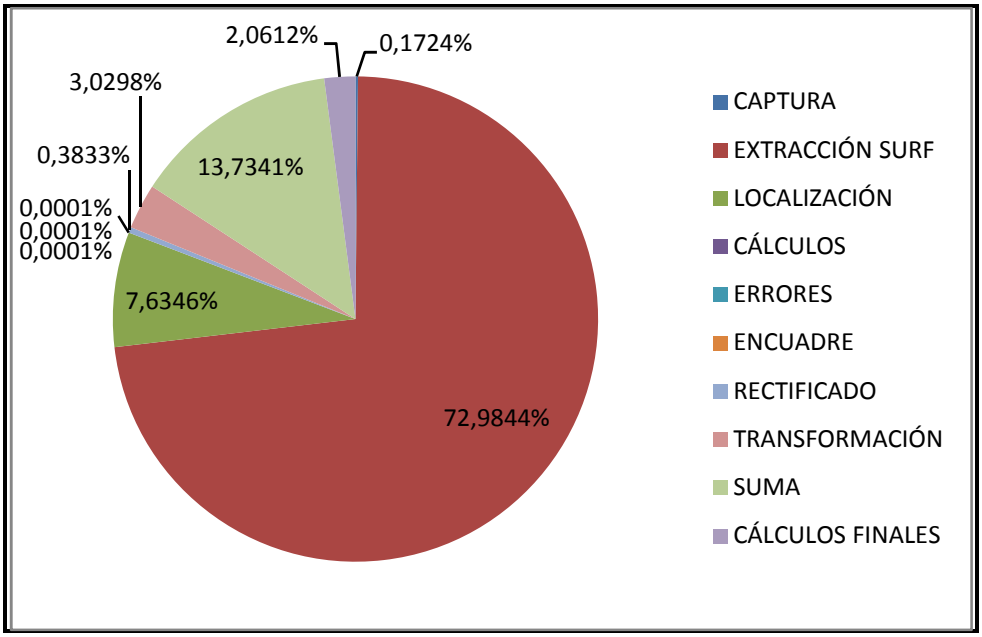


Figura A5: Porcentaje de tiempo medio usado por cada función respecto al tiempo de ejecución total, para la tercera panorámica estudiada.

ESTUDIO POR BLOQUES DE EJECUCIÓN

TIEMPOS DE EJECUCIÓN POR BLOQUES (ms)					TAMAÑO (PÍXELES)
CAPTURA	ANÁLISIS	TRANSF.	SUMA	TOTAL	
2,7309	569,14	20,643	79,394	634,01	307,2
2,8631	506,51	26,36	98,278	746,58	385,25
3,7657	599,59	27,821	115,41	984,27	479,814
1,3586	786,59	35,383	160,93	1215,7	588
1,3716	1003,8	38,202	172,35	1421,4	817,71
1,4469	1182,9	41,746	195,29	1518,2	880,76
1,4473	1243,1	48,417	225,24	1612,5	1003,472
1,415	1316,6	49,749	244,74	1723,8	1134,688
3,5336	1351,6	62,155	306,52	1863,2	1259,296
1,4328	1430,2	72,463	359,12	634,01	1578
					1809,966
MEDIA	2,1366	998,99	42,294	195,73	1239,1
%	0,0017	0,8062	0,0341	0,158	

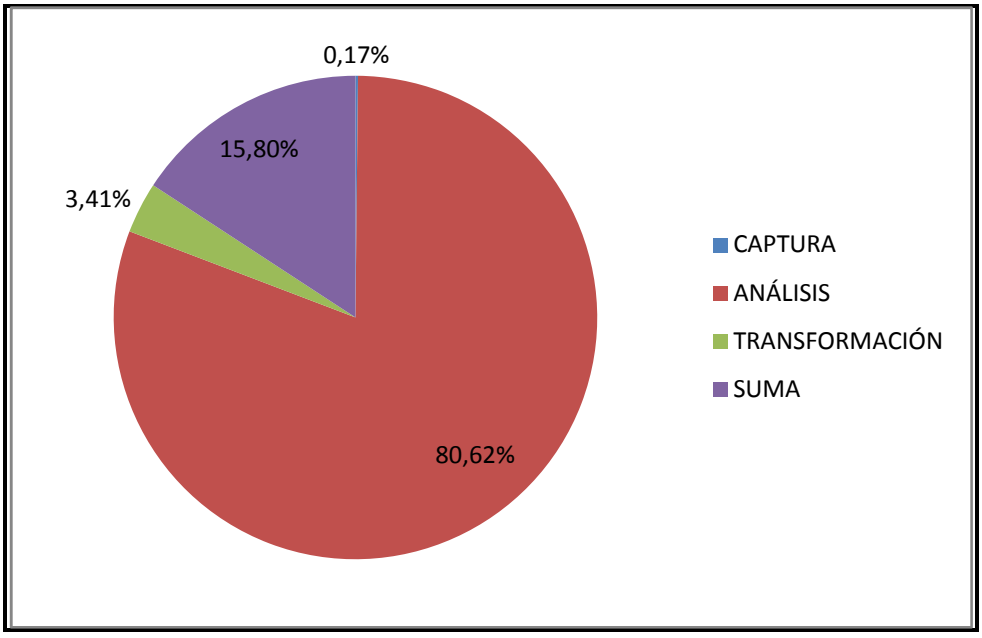


Figura A6: Porcentaje de tiempo medio usado por cada bloque de ejecución del programa respecto al tiempo de ejecución total, para la tercera panorámica estudiada.

PANORÁMICA 4

ESTUDIO POR FUNCIONES DEL PROGRAMA

TIEMPOS DE EJECUCIÓN POR FUNCIÓN (ms)											TAMAÑO (PÍXELES)
INIC.	CAPT.	SURF	LOCAL.	CÁLC.	ERROR	ENC.	RECT.	TRANS.	SUMA	C.FINAL	
388,25	0	0	0	0	0	0	0	0	0	0	0
0	2,8605	898,89	150,85	0,0012	0,0008	0,0011	1,5966	20,932	77,704	13,757	307,2
0	1,4399	1006,4	145,02	0,0012	0,0007	0,0009	2,328	23,526	88,078	14,169	444,106
0	1,4868	1061,4	151,23	0,0011	0,0007	0,0009	3,0328	34,192	130,18	20,275	521,181
0	2,3764	1291,3	177,41	0,001	0,0006	0,001	4,079	35,238	152,16	23,254	754,35
0	1,3616	1419,5	191,91	0,0013	0,0006	0,0008	4,9149	38,335	170,5	26,315	903,08
0	1,3598	1558,6	212,54	0,001	0,0006	0,001	5,6408	39,169	183,8	27,088	1007,94
0	1,4197	1700,2	226,35	0,0009	0,0006	0,001	6,3169	43,043	209,43	30,544	1091,778
0	3,5539	1879,2	257,08	0,0011	0,0008	0,001	7,0374	45,106	213	31,466	1241,556
0	2,3214	1887,5	253,54	0,0011	0,0008	0,001	7,0998	46,237	217,43	34,994	1241,556
0	1,4379	1881,2	242,34	0,001	0,0006	0,0011	6,9882	47,003	238,62	35,386	1275,468
											1414,93
MEDIA	1,9618	1458,4	200,83	0,0011	0,0007	0,001	4,9034	37,278	168,09	25,725	
%	0,001	0,7687	0,1059	6E-07	4E-07	5E-07	0,0026	0,0196	0,0886	0,0136	

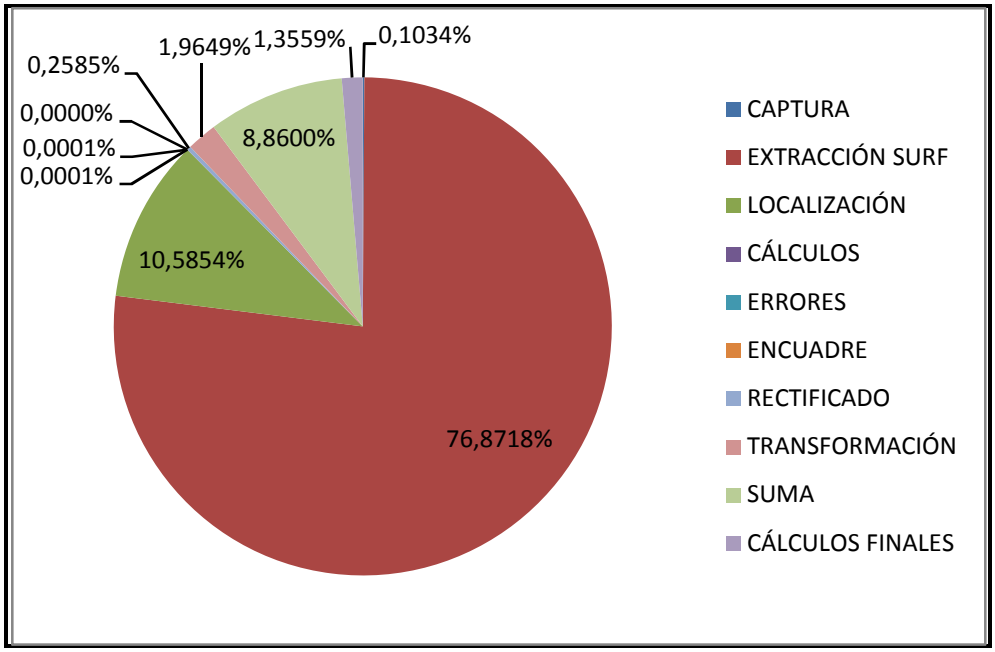


Figura A7: Porcentaje de tiempo medio usado por cada función respecto al tiempo de ejecución total, para la cuarta panorámica estudiada.

ESTUDIO POR BLOQUES DE EJECUCIÓN

TIEMPOS DE EJECUCIÓN POR BLOQUES (ms)					TAMAÑO (PÍXELES)
CAPTURA	ANÁLISIS	TRANSF.	SUMA	TOTAL	
2,8605	1049,7	22,53	91,461	1166,6	307,2
1,4399	1151,4	25,855	102,25	1280,9	444,106
1,4868	1212,6	37,225	150,46	1401,8	521,181
2,3764	1468,7	39,318	175,41	1685,8	754,35
1,3616	1611,4	43,251	196,82	1852,8	903,08
1,3598	1771,1	44,81	210,88	2028,2	1007,94
1,4197	1926,5	49,361	239,97	2217,3	1091,778
3,5539	2136,2	52,144	244,47	2436,4	1241,556
2,3214	2141,1	53,338	252,43	2449,1	1241,556
1,4379	2123,6	53,993	274,01	2453	1275,468
					1414,93
MEDIA	1,9618	1659,2	42,182	193,82	1897,2
%	0,001	0,8746	0,0222	0,1022	

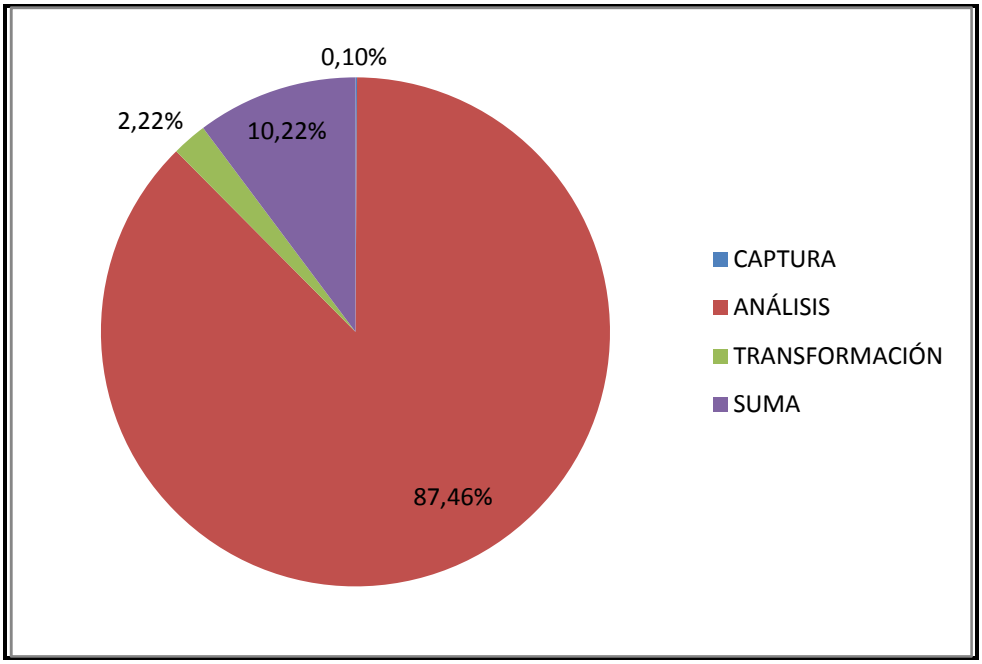


Figura A8: Porcentaje de tiempo medio usado por cada bloque de ejecución del programa respecto al tiempo de ejecución total, para la cuarta panorámica estudiada.

PANORÁMICA 5

ESTUDIO POR FUNCIONES DEL PROGRAMA

TIEMPOS DE EJECUCIÓN POR FUNCIÓN (ms)											TAMAÑO (PÍXELES)
INIC.	CAPT.	SURF	LOCAL.	CÁLC.	ERROR	ENC.	RECT.	TRANS.	SUMA	C.FINAL	
348,95	0	0	0	0	0	0	0	0	0	0	0
0	2,3073	406,05	42,75	0,0012	0,0006	0,001	1,677	21,151	79,318	13,195	307,2
0	2,6476	550,35	58,286	0,0009	0,0006	0,001	2,5134	33,284	114,81	18,112	472,535
0	1,3898	726,28	79,477	0,001	0,0007	0,0011	3,7958	39,426	151,92	24,083	669,579
0	1,3867	918,35	98,159	0,0012	0,0006	0,001	5,4049	66,908	295,38	42,29	884,709
0	2,9357	1775,5	108,07	0,0013	0,0006	0,0011	9,9565	104,31	494,73	71,519	1701,672
0	1,4036	2095,9	95,822	0,001	0,0006	0,0009	16,342	117,58	577,55	78,865	2862,695
0	1,5122	2329,1	100,11	0,0011	0,0006	0,0009	20,645	125,14	683,92	92,694	3381,015
0	1,427	2221,8	98,691	0,001	0,0006	0,0009	22,988	119,83	683,5	88,771	3993,762
0	1,4403	2271,3	108,38	0,001	0,0006	0,001	23,464	136,47	760,58	101,48	3993,762
0	1,5974	2436,3	100,83	0,0011	0,0006	0,001	25,27	150,8	868,36	114,97	4433,988
											5066,776
MEDIA	1,8048	1573,1	89,057	0,0011	0,0006	0,001	13,206	91,49	471,01	64,598	
%	0,0008	0,6827	0,0386	5E-07	3E-07	4E-07	0,0057	0,0397	0,2044	0,028	

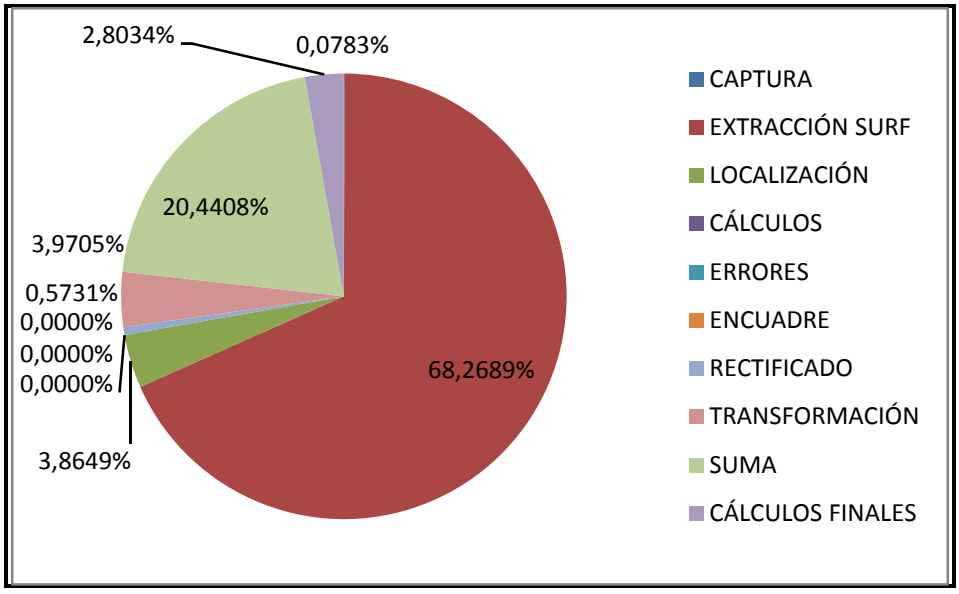


Figura A9: Porcentaje de tiempo medio usado por cada función respecto al tiempo de ejecución total, para la quinta panorámica estudiada.

ESTUDIO POR BLOQUES DE EJECUCIÓN

TIEMPOS DE EJECUCIÓN POR BLOQUES (ms)					TAMAÑO (PÍXELES)
CAPTURA	ANÁLISIS	TRANSF.	SUMA	TOTAL	
2,3073	448,8	22,829	92,513	566,45	307,2
2,6476	608,64	35,798	132,93	780,01	472,535
1,3898	805,76	43,223	176,01	1026,4	669,579
1,3867	1016,5	72,314	337,67	1427,9	884,709
2,9357	1883,5	114,26	566,25	2567	1701,672
1,4036	2191,7	133,93	656,42	2983,5	2862,695
1,5122	2429,2	145,79	776,61	3353,1	3381,015
1,427	2320,5	142,82	772,27	3237	3993,762
1,4403	2379,7	159,94	862,06	3403,1	3993,762
1,5974	2537,1	176,07	983,33	3698,1	4433,988
					5066,776
MEDIA	1,8048	1662,1	104,7	535,6	2304,3
%	0,0008	0,7213	0,0454	0,2324	

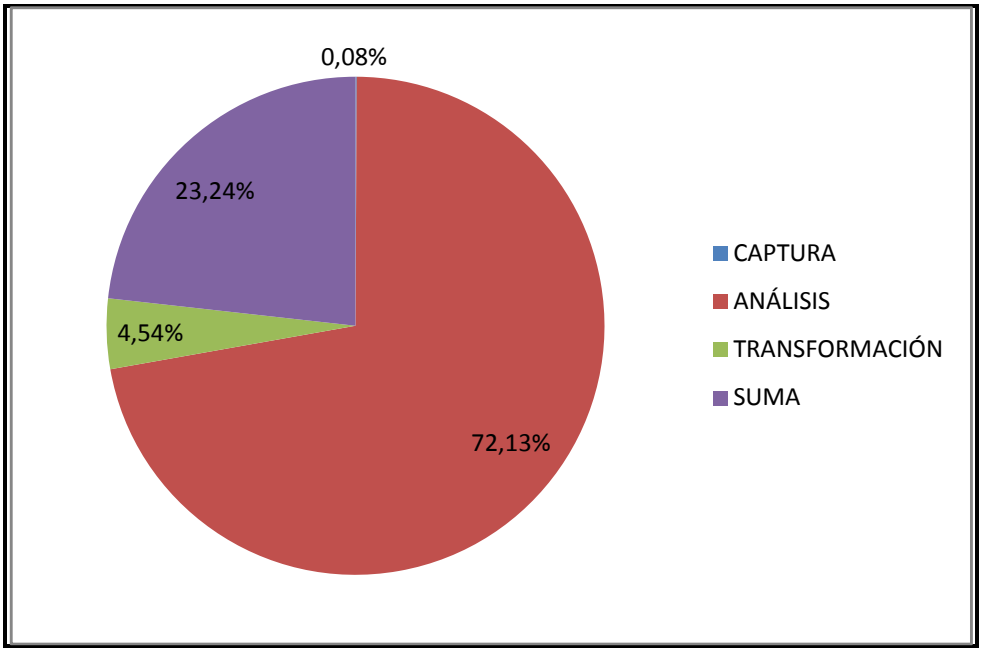
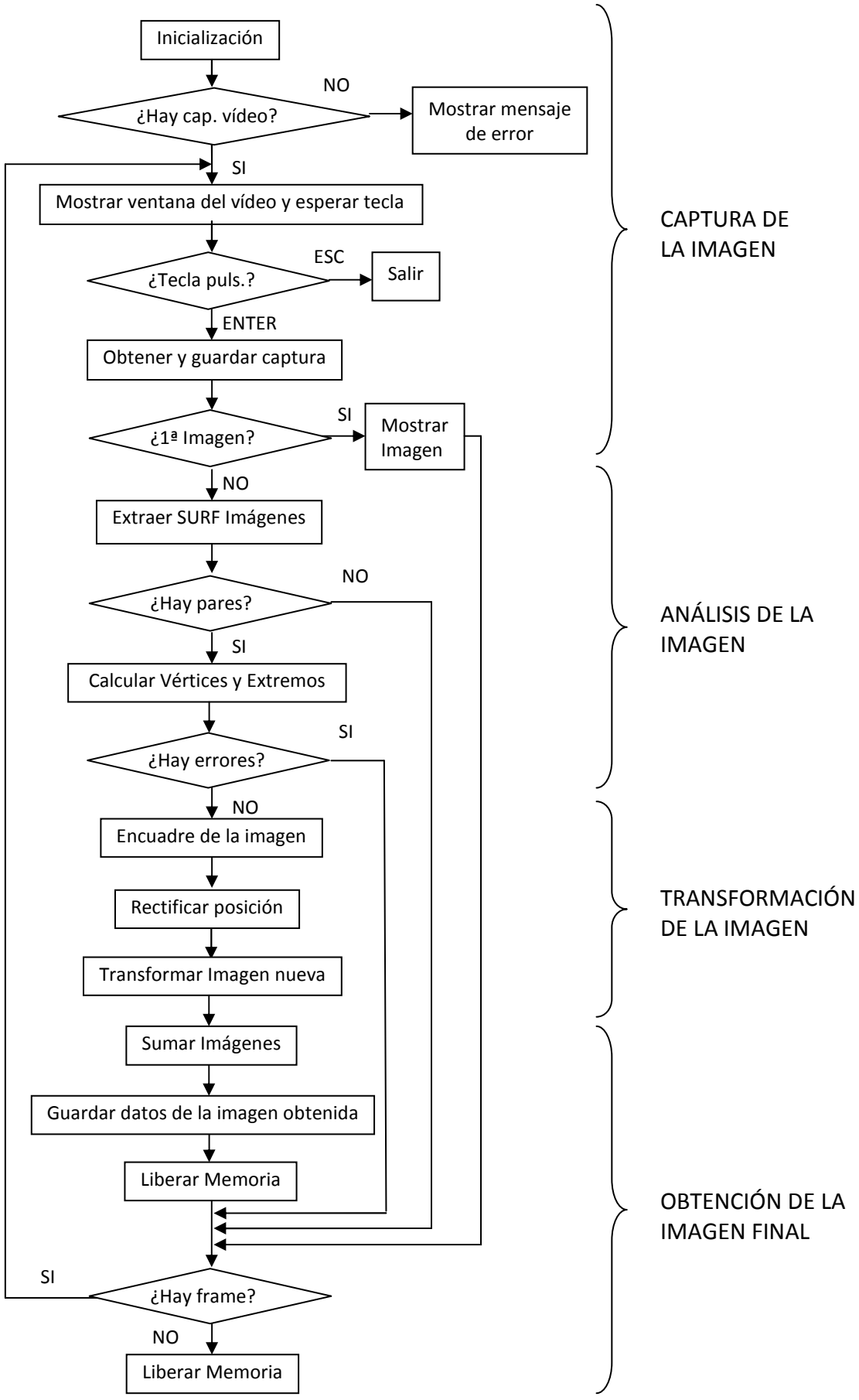


Figura A10: Porcentaje de tiempo medio usado por cada bloque de ejecución del programa respecto al tiempo de ejecución total, para la quinta panorámica estudiada.

ANEXO 2: DIAGRAMA DE FLUJO



ANEXO 3: PROGRAMA

```
////////////////////////////////////////////////////////////////////////////
//                                                                        //
//     PROGRAMA PARA LA CREACIÓN DE MAPAS PANORÁMICOS A PARTIR DE LAS     //
//                    IMÁGENES OBTENIDAS DE UNA CÁMARA WEB                //
//                                                                        //
//                        AUTOR: FERNANDO ORTIZ RENILLA                   //
//                        UNIVERSIDAD CARLOS III DE MADRID                //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

#ifdef _CH_
#pragma package <opencv>
#endif

#define USE_FLANN                       //USO DE LA LIBRERÍA FLANN
#define CV_NO_BACKWARD_COMPATIBILITY

#ifndef _EiC
#include <cv.h>
#include <highgui.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <vector>
#include <algorithm>
#endif

using namespace std;


////////////////////////////////////////////////////////////////////////////
////              FUNCIÓN QUE COMPARA LOS DESCRIPTORES SURF            ////

double compareSURFDescriptors( const float* d1, const float* d2, double best,
                               int length )
{
    double total_cost = 0;
    assert( length % 4 == 0 );
    for( int i = 0; i < length; i += 4 )
    {
        double t0 = d1[i] - d2[i];
        double t1 = d1[i+1] - d2[i+1];
        double t2 = d1[i+2] - d2[i+2];
        double t3 = d1[i+3] - d2[i+3];
        total_cost += t0*t0 + t1*t1 + t2*t2 + t3*t3;
        if( total_cost > best )
            break;
    }
    return total_cost;
}
```

```
////////////////////////////////////////////////////////////////////////
////              FUNCIÓN QUE LOCALIZA LOS VECINOS MÁS CERCANOS       ////

int naiveNearestNeighbor( const float* vec, int laplacian, const CvSeq*
                          model_keypoints, const CvSeq* model_descriptors )
{
    int length = (int)(model_descriptors->elem_size/sizeof(float));
    int i, neighbor = -1;
    double d, dist1 = 1e6, dist2 = 1e6;
    CvSeqReader reader, kreader;
    cvStartReadSeq( model_keypoints, &kreader, 0 );
    cvStartReadSeq( model_descriptors, &reader, 0 );

    for( i = 0; i < model_descriptors->total; i++ )
    {
       const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
       const float* mvec = (const float*)reader.ptr;
       CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
       CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );
       if( laplacian != kp->laplacian )
            continue;
       d = compareSURFDescriptors( vec, mvec, dist2, length );
       if( d < dist1 )
       {
            dist2 = dist1;   dist1 = d;     neighbor = i;
       }
       else if ( d < dist2 )dist2 = d;
    }
    if ( dist1 < 0.6*dist2 )
       return neighbor;
    return -1;
}

////////////////////////////////////////////////////////////////////////
////          FUNCIÓN PARA ENCONTRAR LOS PARES DE PUNTOS CLAVE        ////

void findPairs( const CvSeq* objectKeypoints, const CvSeq* objectDescriptors,
              const CvSeq* imageKeypoints, const CvSeq* imageDescriptors,
              vector<int>& ptpairs )
{
    int i;
    CvSeqReader reader, kreader;
    cvStartReadSeq( objectKeypoints, &kreader );
    cvStartReadSeq( objectDescriptors, &reader );
    ptpairs.clear();

    for( i = 0; i < objectDescriptors->total; i++ )
    {
        const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
        const float* descriptor = (const float*)reader.ptr;
        CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
        CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );
        int nearest_neighbor = naiveNearestNeighbor( descriptor,
                          kp->laplacian, imageKeypoints, imageDescriptors );
        if( nearest_neighbor >= 0 )
        {
            ptpairs.push_back(i);
            ptpairs.push_back(nearest_neighbor);}
    }
}
```

```
////////////////////////////////////////////////////////////////////////
////   FUNCIÓN PARA ENCONTRAR LAS PAREJAS DE PUNTOS CLAVE USANDO FLANN ////

void flannFindPairs( const CvSeq*, const CvSeq* objectDescriptors, const
              CvSeq*, const CvSeq* imageDescriptors, vector<int>& ptpairs )
{
    int length = (int)(objectDescriptors->elem_size/sizeof(float));

    cv::Mat m_object(objectDescriptors->total, length, CV_32F);
    cv::Mat m_image(imageDescriptors->total, length, CV_32F);


    //COPIA DE DESCRIPTORES
    CvSeqReader obj_reader;
    float* obj_ptr = m_object.ptr<float>(0);
    cvStartReadSeq( objectDescriptors, &obj_reader );

    for(int i = 0; i < objectDescriptors->total; i++ )
    {
        const float* descriptor = (const float*)obj_reader.ptr;
        CV_NEXT_SEQ_ELEM( obj_reader.seq->elem_size, obj_reader );
        memcpy(obj_ptr, descriptor, length*sizeof(float));
        obj_ptr += length;
    }

    CvSeqReader img_reader;
    float* img_ptr = m_image.ptr<float>(0);
    cvStartReadSeq( imageDescriptors, &img_reader );

    for(int i = 0; i < imageDescriptors->total; i++ )
    {
        const float* descriptor = (const float*)img_reader.ptr;
        CV_NEXT_SEQ_ELEM( img_reader.seq->elem_size, img_reader );
        memcpy(img_ptr, descriptor, length*sizeof(float));
        img_ptr += length;
    }

    // ENCONTRAR LOS VECINOS MAS CERCANOS USANDO EL ALGORITMO FLANN
    cv::Mat m_indices(objectDescriptors->total, 2, CV_32S);
    cv::Mat m_dists(objectDescriptors->total, 2, CV_32F);
    cv::flann::Index flann_index(m_image, cv::flann::KDTreeIndexParams(4));
    // using 4 randomized kdtrees

    flann_index.knnSearch(m_object, m_indices, m_dists, 2,
                          cv::flann::SearchParams(64));
    // maximum number of leafs checked

    int* indices_ptr = m_indices.ptr<int>(0);
    float* dists_ptr = m_dists.ptr<float>(0);

    for (int i=0;i<m_indices.rows;++i) {
       if (dists_ptr[2*i]<0.6*dists_ptr[2*i+1]) {
          ptpairs.push_back(i);
          ptpairs.push_back(indices_ptr[2*i]);
       }
    }
}
```

```
////////////////////////////////////////////////////////////////////////
////            FUNCION PARA LOCALIZAR OBJETOS PLANOS                ////

int locatePlanarObject( const CvSeq* objectKeypoints, const CvSeq*
    objectDescriptors, const CvSeq* imageKeypoints, const CvSeq*
    imageDescriptors,const CvPoint src_corners[4], CvPoint dst_corners[4],
    const CvPoint res[4] )
{

    double h[9];
    CvMat _h = cvMat(3, 3, CV_64F, h);
    vector<int> ptpairs;
    vector<CvPoint2D32f> pt1, pt2;
    CvMat _pt1, _pt2;
    int i, n;


#ifdef USE_FLANN
    flannFindPairs( objectKeypoints, objectDescriptors, imageKeypoints,
                imageDescriptors, ptpairs );

#else
    findPairs( objectKeypoints, objectDescriptors, imageKeypoints,
                imageDescriptors, ptpairs );

#endif

    n = ptpairs.size()/2;
    if( n < 15 )
       return 0;

    pt1.resize(n);
    pt2.resize(n);

    for( i = 0; i < n; i++ )
    {
      pt1[i]=((CvSURFPoint*)cvGetSeqElem(objectKeypoints,ptpairs[i*2]))->pt;
      pt2[i]=((CvSURFPoint*)cvGetSeqElem(imageKeypoints,ptpairs[i*2+1]))->pt;
    }

    _pt1 = cvMat(1, n, CV_32FC2, &pt1[0] );
    _pt2 = cvMat(1, n, CV_32FC2, &pt2[0] );
    if( !cvFindHomography( &_pt1, &_pt2, &_h, CV_RANSAC, 5 ))
        return 0;

    for( i = 0; i < 4; i++ )
    {
        double x = res[i].x, y = res[i].y;
        double Z = 1./(h[6]*x + h[7]*y + h[8]);
        double X = (h[0]*x + h[1]*y + h[2])*Z;
        double Y = (h[3]*x + h[4]*y + h[5])*Z;
        dst_corners[i] = cvPoint(cvRound(X), cvRound(Y));
    }

    return 1;
}
```

```
////////////////////////////////////////////////////////////////////////
////                  FUNCION PARA OBTENER LA IMAGEN FINAL            ////

IplImage* sumar(const IplImage* cor1, const IplImage* tras, const CvSize tam)
{
    IplImage* suma = cvCreateImage( tam, 8, 3 );
    /*PUNTOS AUXILIARES*/
    CvScalar s1, s2, sR;

    /*SUMA DE LA IMÁGEN TOTAL*/
    for(int j=0; j<cor1->height; j++) {
       for(int i=0; i<cor1->width; i++) {
           s1= cvGet2D(cor1,j,i);
           s2= cvGet2D(tras,j,i);

           if ((s1.val[0]==0 && s1.val[1]==0 && s1.val[2]==0) &&
               (s2.val[0]==0 && s2.val[1]==0 && s2.val[2]==0))
               //ZONA C
               cvSet2D(suma,j,i,cvScalar(0, 0, 0, 0);

           else
           {

               if ((s1.val[0]==0 && s1.val[1]==0 && s1.val[2]==0) ||
                   (s2.val[0]==0 && s2.val[1]==0 && s2.val[2]==0)){
                   //ZONA B
                   sR.val[0] = s1.val[0]+s2.val[0];
                   sR.val[1] = s1.val[1]+s2.val[1];
                   sR.val[2] = s1.val[2]+s2.val[2];
               }

               else {
                   //ZONA A
                   sR.val[0] = s1.val[0]*0.5+s2.val[0]*0.5;
                   sR.val[1] = s1.val[1]*0.5+s2.val[1]*0.5;
                   sR.val[2] = s1.val[2]*0.5+s2.val[2]*0.5;
               }
               cvSet2D(suma,j,i,sR);
           }
       }

    }
    return suma;
}

////////////////////////////////////////////////////////////////////////
////          FUNCIÓN PARA OBTEENER LOS EXTREMOS DE LA IMAGEN         ////

void encuadrar(const int minX, const int maxX, const int minY, const int
    maxY, const CvPoint src_corners[4], const CvPoint dst_corners[4], CvPoint
    esq[4])
{

    for (int c=0; c<4; c++) {
       if (c==0||c==3) {if (esq[c].x > minX)  esq[c].x = minX;}
       if (c==0||c==1) {if (esq[c].y > minY)  esq[c].y = minY;}
       if (c==1||c==2) {if (esq[c].x < maxX)  esq[c].x = maxX;}
       if (c==2||c==3) {if (esq[c].y < maxY)  esq[c].y = maxY;}
    }
```

```
    if (esq[0].x < esq[3].x) esq[3].x = esq[0].x; else esq[0].x = esq[3].x;
     //MÍNIMO X
    if (esq[1].x > esq[2].x) esq[2].x = esq[1].x; else esq[1].x = esq[2].x;
     //MÁXIMO X
    if (esq[0].y < esq[1].y) esq[1].y = esq[0].y; else esq[0].y = esq[1].y;
     //MÍNIMO Y
    if (esq[2].y > esq[3].y) esq[3].y = esq[2].y; else esq[2].y = esq[3].y;
     //MÁXIMO Y
}

////////////////////////////////////////////////////////////////////////
////         FUNCIÓN PARA RECTIFICAR LA POSICIÓN DE LA IMAGEN          ////

IplImage* rectificado(IplImage* cor1, const CvSize tam, const CvPoint
       dst_corners[4], CvPoint esq[4], const int minX, const int minY,
       CvPoint2D32f esq_dst[4])
{
    /*CAMBIO DE TAMAÑO DE CORRESPONDENCIA 1 A tam*/
    IplImage* aux1 = cvCreateImage( cvGetSize(cor1), 8, 3 );
    cvCopy(cor1,aux1 );
    cvReleaseImageHeader(&cor1);
    cor1 = cvCreateImage( tam, 8, 3 );
    //SI TIENE VALORES NEGATIVOS SE MUEVE LA IMAGEN HACIA LA DERECHA
    //Y SE CAMBIAN EL DESTINO DE LA TRANSFORMACIÓN

    if (minX >= 0 && minY >= 0){    //VALORES X E Y POSITIVOS
       cvSetImageROI( cor1, cvRect( 0, 0, aux1->width, aux1->height ) );}

    if (minX < 0 && minY >= 0){     //VALOR X NEGATIVO E Y POSITIVO
       cvSetImageROI( cor1, cvRect( 0-minX, 0, aux1->width, aux1->height ) );
       for (int s=0; s<4; s++){
           esq_dst[s].x = (dst_corners[s].x - minX);
           esq[s].x = esq[s].x - minX;
       }
    }

    if (minX >= 0 && minY < 0){     //VALOR X POSITIVO E Y NEGATIVO
       cvSetImageROI( cor1, cvRect( 0, 0-minY, aux1->width, aux1->height ) );
       for (int s=0; s<4; s++){
           esq_dst[s].y = (dst_corners[s].y - minY);
           esq[s].y = esq[s].y - minY;
       }
    }

    if (minX < 0 && minY < 0){      //VALORES X E Y NEGATIVOS
       cvSetImageROI(cor1,cvRect( 0-minX, 0-minY, aux1->width,aux1->height));
       for (int s=0; s<4; s++){
           esq_dst[s].x = (dst_corners[s].x - minX);
           esq_dst[s].y = (dst_corners[s].y - minY);
           esq[s].x = esq[s].x - minX;
           esq[s].y = esq[s].y - minY;
       }
    }

    cvCopy( aux1, cor1 );
    cvResetImageROI( cor1 );
    cvReleaseImage(&aux1);

    return cor1;
}
```

```
////////////////////////////////////////////////////////////////////////
////        FUNCIÓN PARA RECTIFICAR LA POSICIÓN DE LA IMAGEN EN B/N    ////

IplImage* rectificadoBN(IplImage* bn1, const CvSize tam, const int minX,
                        const int minY)
{
    /*CAMBIO DE TAMAÑO DE CORRESPONDENCIA 1 A tam*/

    IplImage* aux1 = cvCreateImage( cvGetSize(bn1), 8, 1 );
    cvCopy(bn1,aux1 );
    cvReleaseImageHeader(&bn1);
    bn1 = cvCreateImage( tam, 8, 1 );

    //SI TIENE VALORES NEGATIVOS SE MUEVE LA IMAGEN HACIA LA DERECHA
    if (minX >= 0 && minY >= 0)            //TODOS LOS VALORES POSITIVOS
       cvSetImageROI( bn1, cvRect( 0, 0, aux1->width, aux1->height ) );
    if (minX < 0 && minY >= 0)             //VALORES DE X NEGATIVOS
       cvSetImageROI( bn1, cvRect( 0-minX, 0, aux1->width, aux1->height ) );
    if (minX >= 0 && minY < 0)             //VALORES DE Y NEGATIVOS
       cvSetImageROI( bn1, cvRect( 0, 0-minY, aux1->width, aux1->height ) );
    if (minX < 0 && minY < 0)              //VALORES DE X E Y NEGATIVOS
       cvSetImageROI(bn1,cvRect(0-minX, 0-minY, aux1->width, aux1->height));

    cvCopy( aux1, bn1 );
    cvResetImageROI( bn1 );
    cvReleaseImage(&aux1);

    return bn1;
}


////////////////////////////////////////////////////////////////////////
////        FUNCIÓN PARA COMPROBAR POSIBLE ERROR EN EL CÁLCULO        ////

int comprobar_errores(const CvPoint2D32f dst[4],const int anchoG, const int
                      altoG, const int ancho, const int alto )
{

    if (dst[1].x<=dst[0].x || dst[2].y<=dst[1].y || dst[2].x<=dst[3].x ||
          dst[3].y<=dst[0].y || anchoG>(ancho*1.8) || altoG>(alto*2)){

       printf("\nERROR DE CÁLCULO / IMÁGEN DEMASIADO GRANDE\n");
       return 0;}

    else
       return 1;
}


////////////////////////////////////////////////////////////////////////
////            FUNCIÓN PARA TRANSFORMAR LA IMAGEN NUEVA              ////

IplImage* transformar(const CvSize tam, const CvPoint2D32f origen[4], const
                      CvPoint2D32f destino[4], const  IplImage* im1)
{

    /*MATRIZ PARA LA TRANSFORMACIÓN*/
    CvMat *m_tras1;
    m_tras1= cvCreateMat(3, 3, CV_64FC1);

    IplImage* tras = cvCreateImage( tam, 8, 3 );
```

```
    /*TRANSFORMACIÓN DE LA IMAGEN 2*/
    cvGetPerspectiveTransform( origen, destino, m_tras1 );
    // CALCULA LA MATRIZ DE TRASLACIÓN

    cvWarpPerspective(im1, tras, m_tras1 );
    // TRANSFORMA LA IMAGEN SEGÚN LA MATRIZ

    return tras;
}


////////////////////////////////////////////////////////////////////
////                          MAIN                              ////

int main()
{

    ////*----DECLARACIONES DE VARIABLES----*////

    char Vid[] = "WebCam";
    IplImage *frm;
    CvCapture *capture;
    capture=cvCaptureFromCAM(1);    //0 PARA WEBCAM POR DEFECTO
                                    //1 PARA WEBCAM USB

    /*RESOLUCIÓN DE LA WEBCAM:    640 X 480 PÍXELES*/

    CvPoint res[4] = {{0,0}, {640,0}, {640,480}, {0, 480}};
    CvPoint2D32f res_32f[4];
    for (int n=0; n<4; n++)
       res_32f[n] = cvPointTo32f (res[n]);

    CvSize dim;
    dim.height=480;
    dim.width=640;

    /*IMÁGENES*/

    IplImage* im1 = cvCreateImage( dim, 8, 1 );
    //IMAGEN CAPTURA DE LA CÁMARA EN B/N

    IplImage* col1 = cvCreateImage( cvGetSize(im1), 8, 3 );
    //IMAGEN CAPTURA DE LA CÁMARA EN COLOR

    IplImage* corBN = cvCreateImage( cvGetSize(im1), 8, 1 );
    //IMAGEN RESULTANTE EN B/N

    IplImage* cor1 = cvCreateImage( cvGetSize(im1), 8, 3 );
    //IMAGEN RESULTANTE EN COLOR

    /*VARIABLE DE CONTROL DE LA PRIMERA IMAGEN*/
    int primeraimagen=1;            //ACTIVADO / DESACTIVADO

    /*ESQUINAS DE DESTINO DE LA TRANSFORMACIÓN*/
    CvPoint2D32f esq_dst[4];

    /*PARÁMETROS DE ALMACENAMIENTO*/
    CvMemStorage* storage = cvCreateMemStorage(0);
    CvSURFParams params = cvSURFParams(500, 1);
```

```
/*ESQUINAS DE ORIGEN*/
CvPoint src[4] = {{0,0}, {im1->width,0}, {im1->width, im1->height},
          {0, im1->height}};

/*ESQUINAS DE DESTINO*/
CvPoint dst[4] = {src[0],src[1],src[2],src[3]};

/*PUNTOS DE LAS ESQUINAS DE LA NUEVA IMAGEN GRANDE*/
CvPoint esq[4] = {{0,0}, {640,0}, {640,480}, {0, 480}};

/*VALORES MÍNIMOS EN X E Y (PARA TESTEAR VALORES NEGATIVOS)*/
int minX, minY, maxX, maxY;



////*----CAPTURA DE LA IMAGEN----*////

if(capture){

    /*VENTANA DE PRESENTACIÓN*/
    cvNamedWindow ("Correspondencia", 0);
    cvNamedWindow(Vid,1);

    do{
        /*PUNTOS CLAVE Y DESCRIPTORES*/
        CvSeq *im1Pclave = 0, *im1Des = 0;
        CvSeq *im2Pclave = 0, *im2Des = 0;

        /*SE OBTIENE EL FRAME*/
        frm=cvQueryFrame(capture);

        if(frm)
            /*VENTANA DEL VÍDEO*/
            cvShowImage (Vid,frm);

        int tecla=cvWaitKey(50);     //ESPERAMOS A QUE SE PULSE UNA TECLA

        if (tecla==27) break;        //ESC PARA SALIR
        if (tecla==10)               //ENTER PARA FOTO
        {
            /*GUARDAMOS EL FRAME EN COL1*/
            col1 = frm;
            cvCvtColor( col1, im1, CV_BGR2GRAY );
            //COL1 SE CONVIERTE A B/N EN IM1

            /*SI ES LA PRIMERA IMAGEN SE MUESTRA DIRECTAMENTE*/
            if( primeraimagen == 1)
            {
                primeraimagen = 0;
                cvCopy(col1, cor1);
                cvCvtColor( cor1, corBN, CV_BGR2GRAY );
                cvShowImage( "Correspondencia", cor1);
            }

            /*SI NO ES LA PRIMERA IMAGEN SE CONTINUA EL ALGORITMO*/
            else{



            ////*----ANÁLISIS DE LAS IMÁGENES----*////
```

```
/*EXTRAER PUNTOS CLAVE DE IM1 (IMAGEN NUEVA)*/
cvExtractSURF(im1,0, &im2Pclave, &im2Des, storage, params);

/*EXTRAER PUNTOS CLAVE DE COR (IMAGEN PANORAMICA)*/
cvExtractSURF( corBN, 0, &im1Pclave, &im1Des, storage,
               params);

/*LOCALIZAR LA FOTO. CREAR PUNTOS DE DESTINO DE LA
TRANSFORMACIÓN*/
if(locatePlanarObject(im2Pclave, im2Des, im1Pclave, im1Des,
       src, dst, res))
{
    for (int s=0; s<4; s++)
        esq_dst[s] = cvPointTo32f (dst[s]);

    /*CÁLCULO DE LOS VALORES MÍNIMOS DE X E Y*/
    minX= min( min (src[0].x, dst[0].x), min (src[3].x,
           dst[3].x));
    minY= min( min (src[0].y, dst[0].y), min (src[1].y,
           dst[1].y));
    maxX= max( max (src[1].x, dst[1].x), max (src[2].x,
           dst[2].x));
    maxY= max( max (src[2].y, dst[2].y), max (src[3].y,
           dst[3].y));

    /*ANCHO Y ALTO DE LA IMAGEN PEQUEÑA*/
    int ancho, alto;
    ancho = cor1->width;
    alto = cor1->height;

    /*ANCHO Y ALTO DE LA IMAGEN GRANDE*/
    int anchoG, altoG;
    anchoG = abs(minX) + abs(maxX);
    altoG = abs(minY) + abs(maxY);

    /*TAMAÑO DE LA IMAGEN GRANDE*/
    CvSize tam;
    tam.height=altoG;
    tam.width=anchoG;

    if (comprobar_errores(esq_dst,anchoG,altoG,ancho, alto))
        {

        ////*----TRANSFORMACIÓN DE LA IMAGEN----*////

        /*A PARTIR DE ESTOS EXTREMOS SE CREA UN RECTÁNGULO
        QUE CONTENGA LA IMAGEN*/
        encuadrar(minX, maxX, minY, maxY, src, dst, esq);

        /*RECTIFICADO DE POSICIÓN*/
        cor1 = rectificado(cor1, tam, dst, esq, minX, minY,
                           esq_dst);
        corBN = rectificadoBN (corBN, tam, minX, minY);

        /*TRANSFORMACIÓN DE LA IMAGEN 2*/
        IplImage* transformada = cvCreateImage( tam, 8, 3 );
        transformada= transformar(tam,res_32f,esq_dst,col1);
```

```
                        ////*----OBTENCIÓN DE LA IMAGEN FINAL----*////

                        /*IMAGEN TOTAL*/
                        IplImage* suma = cvCreateImage( tam, 8, 3 );
                        IplImage* sumaBN = cvCreateImage( tam, 8, 1 );
                        suma = sumar( cor1, transformada, tam);

                        /*REPRESENTACIÓN DE LA IMAGEN TOTAL*/
                        cvShowImage( "Correspondencia", suma);

                        im1 = cvCreateImage( dim, 8, 1 );
                        col1 = cvCreateImage( dim, 8, 3 );

                        /*GUARDAR IMAGEN*/
                        cvCvtColor( suma, sumaBN, CV_BGR2GRAY );
                        cvCopy(sumaBN, corBN);
                        cvCopy(suma, cor1);

                        /*SE GUARDAN LOS DATOS DE LA IMAGEN OBTENIDA*/
                        for (int s=0; s<4; s++) {
                            src[s].x = esq[s].x;
                            src[s].y = esq[s].y;
                        }

                        /*SE LIBERA LA MEMORIA RESERVADA*/
                        cvReleaseImage(&transformada);
                        cvReleaseImage(&suma);
                        cvReleaseImage(&sumaBN);

                    }   //FIN if(comprobar_errores)

                } //FIN if(locateplanarobject)

                else
                    printf("\nLOCALIZACIÓN IMPOSIBLE: POCAS PAREJAS DE
                            PUNTOS CLAVE\n");

            } //FIN else flag=0

        } //FIN if(k=ENTER)

    } while(frm); //FIN do-while(frm)

    /*SE LIBERA LA MEMORIA RESERVADA*/
    cvDestroyWindow("Correspondencia");
    cvDestroyWindow(Vid);
    cvReleaseCapture (&capture);
    cvReleaseImage(&im1);
    cvReleaseImage(&cor1);
    cvReleaseImage(&col1);
    cvReleaseImage(&corBN);

   } //FIN if(capture)

   else
       printf("\nERROR: NO SE PUEDE ACCEDER A LA WEBCAM\n");

   return 0;
}
```